uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

2018-2019

Trabajo Fin de Grado

Diseño y Desarrollo de un Sistema de Ficheros Distribuido y Paralelo basado en Apache Cassandra

---

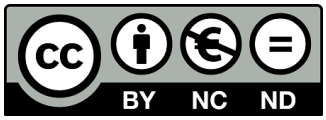
Elías Del Pozo Puñal

Tutor

Félix García Carballeira

Leganés, Madrid, España

Julio 2019



Con frecuencia las dificultades preparan a personas ordinarias para un destino extraordinario.

C. S. Lewis

Diseño y Desarrollo de un Sistema de Ficheros Distribuido y Paralelo basado en Cassandra

por Elías Del Pozo Puñal

Conforme pasan los años, va aumentando la necesidad de almacenar grandes cantidades de información a medida que también va aumentando el tamaño de los datos que los usuarios tienen a su disposición. Para ello, van surgiendo diferentes sistemas de ficheros, bases de datos o sistemas de almacenamiento que permiten a estos usuarios guardar ficheros o información cada vez más grandes. Por ello, Apache Cassandra es una Base de Datos que ofrece la posibilidad de almacenar grandes cantidades de información sin comprometer el rendimiento ni la disponibilidad. Esta Base de Datos ofrece la posibilidad de que pueda implantarse en diferentes nodos permitiendo la comunicación entre todos ellos sin perder la información si alguno de esos nodos falla.

Durante este trabajo se va a proceder a explicar el diseño e implementación de una interfaz para la Base de Datos Apache Cassandra, de tal forma que pueda ser utilizada como un Sistema de Ficheros, basándose en las llamadas de POSIX. De igual forma, se realizará una interfaz para MPI-IO con las mismas funciones. Esta Base de Datos, desarrollada por la Apache Software Foundation, es una Base de Datos no relacional, que se basa en el modelo "clave-valor" como modelo de almacenamiento. Este modelo de almacenamiento es muy similar al concepto de Map-Reduce, técnica que permite el procesado de datos gracias a la generación de tuplas (<clave>, <valor>).

Para poder realizar esta interfaz de llamadas a Cassandra, se utilizará un conector que permitirá la conexión con la BBDD y se procederá a realizar una serie de pruebas que determinarán la capacidad y el rendimiento de Cassandra con diferentes tipos de tamaño de fichero y diferentes clientes, además de una comparación con el rendimiento de otros sistemas de ficheros como, por ejemplo, HDFS [1].

Palabras clave: Apache Cassandra, POSIX, NoSQL, Map-Reduce, HDFS

Supervisor: Félix García Carballeira
Title: Full Professor

Design and Development of a Distributed and Parallel File System based on Apache Cassandra

by Elías Del Pozo Puñal

As the years go by, the need to store large amounts of information increases as the size of the data that users have at their disposal also increases. To this end, different file systems, databases or storage systems are emerging that allow these users to save ever larger files or information. Therefore, Apache Cassandra is a Database that offers the possibility of storing large amounts of information without compromising performance or availability. This Database offers the possibility that it can be implanted in different nodes allowing communication between all of them without losing the information if any of those nodes fail.

During this work will proceed to explain the design and implementation of an interface for the Database Apache Cassandra, so that it can be used as a File System, based on calls from POSIX. Likewise, an interface for MPI-IO with the same functions will be created. This Database, developed by the Apache Software Foundation, is a non-relational Database based on the key-value model as a storage model. This storage model is very similar to the concept of Map-Reduce, a technique that allows data processing thanks to the generation of tuples (<key>, <value>).

In order to be able to make this interface of calls to Cassandra, a connector will be used that will allow the connection with the DDBB and a series of tests will be carried out that will determine the capacity and performance of Cassandra with different types of file size and different clients, as well as a comparison with the performance of other file systems such as, for example, HDFS [1].

Keywords: Apache Cassandra, POSIX, NoSQL, Map-Reduce, HDFS

Supervisor: Félix García Carballeira
Title: Full Professor

Agradecimientos

En esta sección me gustaría agradecer a todas las personas que han estado apoyándome para poder realizar este proyecto.

En primer lugar, se lo quiero agradecer a mis padres, Ángel y Araceli y a mi hermano, Víctor, y en general a toda mi familia, por haberme ayudado y apoyado desde siempre y sobre todo durante estos años que he estado en la Universidad. Sin ellos no hubiera sido posible que yo estuviera realizando este Trabajo de Fin de Grado.

En segundo lugar, quiero agradecérselo a mi tutor Félix por haberme dado la oportunidad de realizar este proyecto, proyecto que ha resultado interesante y que me ha permitido hacer uso de gran parte de los conocimientos que he adquirido durante los años de la carrera y que me permitirán avanzar como Ingeniero Informático.

Por último, pero no menos importante, también quiero acordarme de mis amigos, tanto los que he tenido antes de llegar a la Universidad, como Víctor, como aquellos que he conocido durante estos últimos años y que actualmente estamos pasando los últimos compases de este grado juntos, como Diego, Luis y Noelia, que han podido ayudarme, de una forma u otra, a terminar esta memoria.

A todos ellos, muchas gracias por aguantarme todos estos años.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este primer capítulo se presentará el proyecto que se ha realizado, junto con los motivos por los que se ha decidido realizar la implementación para la realización de la interfaz con la Base de Datos (ver sección 1.1), además de unos objetivos a cumplir durante la realización del proyecto (ver sección 1.2) y, por último, se describirá la estructura del presente documento (ver sección 1.3).

1.1 Motivación y Desarrollo

La cantidad de información con la que se trabaja durante los últimos años ha ido incrementándose con el avance de la tecnología gracias a la aparición de nuevos dispositivos electrónicos y al desarrollo de nuevas tecnologías que permiten crear datos cada vez más grandes. Esto ha provocado que, junto con el aumento de nuevos usuarios que tienen acceso a Internet, haya cada vez una mayor cantidad de datos que se envían entre diferentes puntos de la red a velocidades cada vez mayores. Por estas razones, en los últimos años, se está registrando una producción de 2.5 exabytes de datos por día, provenientes de muchos campos, como puede ser el sector de la salud, del Internet de las cosas o, incluso de la meteorología, entre muchos otros. [2]

Para poder almacenar toda esta información diaria, es necesaria la creación de sistemas de ficheros o de bases de datos que no sigan los métodos tradicionales y no tengan limitaciones en cuanto a capacidad ni a velocidades de cómputo y respuesta. Es por la existencia de este tipo de información que surgió el término del Big Data [3].

Este término hace referencia a un conjunto de datos que generan todo tipo de dispo-

sitivos y que necesitan un tratamiento especial debido al tamaño de la información con la que se trata, de tal forma que es necesario el uso de procedimientos específicos para procesar los datos y, además, en un tiempo admisible. [4]

Una de las primeras empresas que hicieron uso del Big Data [3], y que actualmente es muy usado por la gran mayoría de las personas que hacen uso de Internet, es Google [5]. Esta compañía, conocida por el motor de búsqueda del mismo nombre, desarrolló su propio Sistema de Ficheros [6], que recibe el nombre de Google File System [7][8].

Este Sistema de Ficheros[6] que diseñaron estuvo orientado a manejar las grandes cantidades de información con las que trabajaban en Google para poder aumentar el rendimiento del almacenamiento de datos además del procesamiento de éstos. El diseño de este Sistema de Ficheros les funcionó tan bien que lo implantaron en toda la compañía como plataforma de almacenamiento y de tratamiento de datos haciendo uso de una gran cantidad de nodos con mucho almacenamiento disponible. Más concretamente, poseen miles de discos duros con cientos de terabytes cada uno. Este almacenamiento, además, puede ser usado por miles de personas de forma simultánea, gracias al servicio que ofrece Google llamado Google Drive [9].

Gracias a los resultados que obtuvo la compañía de Google, otras compañías decidieron desarrollar sus propios sistemas de ficheros y Base de Datos enfocados en el campo del Big Data. De entre todas las compañías existentes, la empresa de Software libre Apache Software Foundation [10] desarrolló la Base de Datos Apache Cassandra [11].

Apache Cassandra es una Base de Datos no relacional, es decir, su sistema de gestión de Base de Datos es diferente a lo que se usa de forma tradicional y, además, está orientado al almacenamiento de gran cantidad de datos, ofreciendo a los usuarios la posibilidad de realizar operaciones de baja latencia. También, Cassandra ofrece una alta escalabilidad y rendimiento, permitiendo que el usuario no pierda mucho tiempo a la hora de hacer alguna petición a la Base de Datos.

Entrando en más detalle, con respecto a que Cassandra es una Base de Datos NoSQL [12], lo que se quiere decir es que no sigue el modelo clásico que se estaba utilizando hasta el momento con las bases de datos relacionales, como, por ejemplo, MySQL [13], desarrollado por Sun Microsystems y Oracle Corporation [14]. Cassandra, al utilizar este sistema de gestión de datos, consigue que pueda manejar grandes cantidades de datos, ne-

cesario para el Big Data [3], además de que es muy sencillo escalarlo de forma horizontal, es decir, que resulta menos complicado distribuir el trabajo y aumenta el rendimiento si se añaden más nodos al cluster que conforme la Base de Datos, aunque esto pueda requerir un mayor mantenimiento.

Por otro lado, para poder almacenar toda la información que se necesite en esta Base de Datos, es necesario hacer uso de algún tipo de interfaz que permita el traspaso de datos. Para ello, la empresa DataStax [15] pone a disposición de los usuarios de un conector o controlador en diferentes lenguajes de programación para poder interactuar con Cassandra.

Lo que se quiere conseguir con este proyecto es, con la ayuda del conector que provee la empresa DataStax mencionada anteriormente, diseñar e implementar un Sistema de Ficheros que utilice la Base de Datos Cassandra por debajo para el almacenamiento de datos. Este Sistema de Ficheros estará diseñado en base al estándar POSIX [16] escrita por el IEEE creando diversas funciones importantes para el manejo de ficheros y directorios.

Una vez que el desarrollo de este Sistema de Ficheros [6] se complete, lo que se quiere conseguir es integrarlo de forma parcial en una Interfaz de Paso de Mensajes, conocido como MPI [17], que consiste en una biblioteca que se usa para Sistemas Distribuidos, de tal forma que se permita que varias computadoras puedan ejecutar operaciones simultáneamente. Más concretamente, MPI ofrece además una interfaz de entrada y salida que permite el uso de diversos Sistemas de Ficheros. Esta interfaz se llama MPI-IO [18] y es la interfaz que se usará, teniendo por debajo las funciones que se desarrollen para el manejo de ficheros y directorios sobre Apache Cassandra.

1.2 Objetivos

En este proyecto se muestran diferentes objetivos, de entre los cuales se quiere destacar, de forma más general, el siguiente: **estudio de la aplicación de una Base de Datos NoSQL para su uso como un Sistema de Ficheros Distribuido y Paralelo**. Para poder conseguir este objetivo, se deben realizar en este proyecto los siguientes pasos:

- Diseño e implementación de una biblioteca básica de entrada y salida para poder hacer uso de la Base de Datos Apache Cassandra.

- Implementación de una interfaz basada en POSIX con la biblioteca de entrada y salida para la Base de Datos.
- Implementación de una interfaz basada en MPI-IO con la biblioteca de entrada y salida para la Base de Datos.
- Obtención del rendimiento ofrecido por la biblioteca desarrollada junto con el rendimiento de un Sistema de Ficheros al uso como es Hadoop Distributed File System (HDFS).

1.3 Capítulos

El presente documento estará formado por los siguientes puntos o capítulos:

- Capítulo 1 Introducción: En este capítulo se muestra una breve descripción del trabajo que se ha realizado junto con los objetivos a conseguir.
- Capítulo 2 Estado del arte: En este capítulo se realiza un breve resumen de las tecnologías que se usan actualmente y, además, se detallarán aquellas que han sido usadas en el proyecto.
- Capítulo 3 Análisis: En este capítulo se detallará el problema que da forma a este proyecto, la solución o soluciones elegidas y se definirá un conjunto de requisitos. Por último, se definirá el marco regulatorio del presente proyecto.
- Capítulo 4 Diseño: En este capítulo se mostrará el diseño elegido para la realización del proyecto y de los componentes que lo forman.
- Capítulo 5 Implementación y desarrollo: En este capítulo se definirá el proceso de implementación de la interfaz de acceso y manejo de Cassandra.
- Capítulo 6 Verificación, validación y evaluación: En este capítulo se definirá la verificación y validación del proyecto. Además, se mostrará el rendimiento de la interfaz desarrollada con diferentes pruebas realizadas.
- Capítulo 7 Planificación y Costes: En este capítulo se realizará la planificación que se ha llevado a cabo en el proyecto junto con un presupuesto. Finalmente, se describirá el entorno socio-económico.

- Capítulo 8 Conclusiones y trabajo futuro: Finalmente, en este capítulo se presentarán unas conclusiones y reflexiones que se han alcanzado tras el desarrollo del proyecto y se definen unas guías de cara a investigaciones futuras.

Capítulo 2

Estado del arte

En el presente capítulo se explicará el punto en el que se encuentra la tecnología que ha sido usada para completar este proyecto. Se pretende dar una idea general sobre los Sistemas de Ficheros Distribuidos en primer lugar (ver sección 2.1), seguido de varias características que tiene el Big Data [3] (ver sección 2.2) que, actualmente, es una parte importante en lo referente al manejo de información. Además, como parte fundamental de este proyecto, se hablará en profundidad de la Base de Datos que (ver sección 2.3) se utilizará para desarrollar este sistema utilizando varias de las funciones y métodos que ofrece POSIX (ver sección 2.4). Por último, se explicarán los fundamentos de la Interfaz MPI que será usado sobre la implementación que accederá a Apache Cassandra (ver sección 2.5).

Como se parte del hecho de que lo que se quiere hacer es un Sistema Distribuido, conviene explicar de antemano en qué consiste. Un Sistema Distribuido es aquel que está formado por una serie de componentes que están situados en diferentes puntos, como pueden ser computadores, y que están conectados entre ellos por la red, de tal forma que se comunican y se coordinan entre ellos gracias al paso de mensajes sin memoria ni reloj común.[19][20]

Estos Sistemas Distribuidos presentan una serie de características a tener en cuenta a la hora de desarrollar el presente proyecto:

- Permite compartir recursos a nivel de Hardware y Software.
- Tiene una gran capacidad de crecimiento, es decir, tiene una alta escalabilidad.
- Ofrece tolerancia a fallos, esto es, en caso de que algún componente del sistema

falle existen mecanismos de seguridad que permiten que la información almacenada en ese componente se pueda restablecer.

- Permite concurrencia entre procesos, es decir, es capaz de manejar el uso de diversos clientes que acceden a los recursos del sistema a la vez.

No obstante, también tiene una serie de inconvenientes que caben destacar y que se deben tener en cuenta si se quiere utilizar este tipo de sistema:

- Problemas de fiabilidad en la interconexión de los componentes
- Tiene un alto coste de mantenimiento
- Las comunicaciones entre componentes pueden ser inseguras
- Software más complejo

Uno de las características más importantes que se van a tener en cuenta en el proyecto que se está desarrollando es la tasa de transferencia de los datos que va a manejar la Base de Datos elegida, esto es, la velocidad de transferencia de los datos, en comparación con Sistemas de Ficheros que están siendo usados en la actualidad.

2.1 Sistemas de Ficheros Distribuidos

Un tema que se debe considerar antes de realizar el proyecto es comprender lo que es un Sistema de Ficheros [6] y, más concretamente, un Sistema de Ficheros Distribuido. Para empezar, un Sistema de Ficheros es un sistema que permite almacenar y organizar los ficheros en un computador junto con los datos que están contenidos en ellos, además de poder encontrarlos y acceder a ellos con mayor facilidad. Estos sistemas de ficheros se usan en la actualidad en los dispositivos conocidos como discos duros y, también SSD o Discos de Estado Sólido.[20]

En la actualidad, la mayoría de entornos y aplicaciones utilizan estos ficheros como una manera de acopio permanente de la información y añade una nueva forma para compartir estos datos guardados. Estos Sistemas de Ficheros, para poder organizar la información, lo que hacen es dividir los datos en bloques de datos y distribuirlos de forma aleatoria en el dispositivo de almacenamiento para hacer más fácil su accesibilidad.

Con los bloques de datos antes mencionados, cabe decir que se tienen que explicar varias características que serán necesarias a la hora de diseñar el proyecto en el cual la tasa de transferencia juega un papel importante:[19]

- Tiempo de acceso: Es el tiempo que tarda en llegar a un bloque desde que el usuario lo pide al dispositivo de almacenamiento. La idea de este proyecto es conseguir una alta tasa de transferencia para que este tiempo de acceso a bloque se reduzca lo máximo posible, pudiendo manejar ficheros en el menor tiempo posible.

- Tamaño de bloque: Cantidad de bytes de datos que se leerán del fichero o se escribirán al fichero que manejará el Sistema de Ficheros[6]. A mayor tamaño de bloque mayor será la información que se trate pero afectará al tiempo de acceso aumentándolo. Otra cosa a tener en cuenta es que si el tamaño de bloque es muy grande, con ficheros pequeños, se desaprovechará mucho el espacio.

Como ya se ha explicado al principio de este punto lo que es un Sistema Distribuido, es necesario decir que, la combinación de estos dos sistemas, da lugar a los Sistemas de Ficheros Distribuidos, sistemas que sirven para compartir y almacenar ficheros y otros recursos de manera persistente en una red de computadores. Estos sistemas permiten que varios procesos de diferentes computadores puedan acceder a ficheros de manera conjunta[20][21]. A continuación se van a explicar diferentes ejemplos de Sistemas de Ficheros Distribuidos que sirven de ayuda para tener una idea más específica para el desarrollo de este trabajo:

- NFS [21]: Sistema de Ficheros diseñado por Sun Microsystems que permite acceder a los ficheros y a los directorios de forma remota de tal forma que parece que se accede localmente de cara al usuario. Este acceso transparente e independiente del sistema operativo que se esté usando es gracias al uso de llamadas a procedimientos remotos (RPC) entre cliente y servidor, siendo el cliente aquel que pide los datos y el servidor aquel que contiene los datos y los envía al cliente respondiendo la petición. Una característica importante de este Sistema de Ficheros [6] es que el servidor no tiene estado, es decir, no mantiene información sobre la fase en la que se encuentran sus otros clientes para funcionar correctamente.

- GFS [20]: Este Sistema Distribuido permite acceder y compartir discos a diferentes nodos dispuestos en una red de almacenamiento. Para ello, distribuye la carga de trabajo y las obligaciones a través de los diferentes nodos de proceso y el almacenamiento entre los diferentes dispositivos de almacenamiento que posea el sistema.

- Google File System [7][20]: Este Sistema de Ficheros es un sistema de almacenamiento diseñado para satisfacer las propias necesidades de la empresa que lo ha desarrollado, Google, por lo que no es un sistema que esté enfocado para su uso por el público general. Este sistema contiene un nodo maestro que gestiona los metadatos de los ficheros como puede ser datos para controlar el acceso a la información, mapa de la distribución de los datos guardados, el espacio de nombres usado, ubicación de los bloques de datos, entre otros. Como se puede ver, este nodo maestro es el encargado de los metadatos del sistema, mientras que los propios datos se encuentran repartidos en bloques de datos en múltiples servidores. Estos bloques tienen un identificador único asignado por el nodo o servidor maestro además de que, para aumentar la fiabilidad del sistema, se replican estos bloques en varios nodos para evitar su pérdida.

2.2 Big Data

Como se ha dicho anteriormente, Big Data[3] es un término que se usa para conjuntos masivos de datos que se originan de muchos elementos electrónicos, como pueden ser los teléfonos móviles, los ordenadores, el Internet de las cosas, entre otros muchos elementos que actualmente están siendo usados de forma masiva.[4] Debido a esta masificación del uso de la información gracias al avance tecnológico y de las redes es necesario que se manejen tiempos de cómputo lo más corto posibles para poder ofrecer a los usuarios una rápida respuesta. Es por ello que las Bases de Datos relacionales que existen actualmente, tales como Oracle[14], MariaDB[22] o MySQL[13] no sirven para procesar esta cantidad de datos en un tiempo aceptable.

Dada la importancia que tiene el Big Data hoy en día, es necesario tener en cuenta que posee varias características que provocan que surja la necesidad de tratar esta información cuanto antes y de la mejor forma posible[23]:

- Volumen: Actualmente se están manejando cantidades de datos del orden de petabytes o, incluso, exabytes.
- Velocidad: Debido a que la cantidad de información que se produce es tan grande que es necesario que estos datos se manipulen a altas velocidades, además de que también se debe manejar la información con tecnologías que trabajen en tiempo real
- Variedad: Como la información que se maneja es muy distinta, y, por tanto, heterogénea, se deben realizar acciones que permitan estructurar los datos para mejorar su manipulación
- Veracidad: Otro de los temas más importantes en la actualidad es la seguridad de la información. Actualmente, como para el tratamiento de información se utilizan intermediarios y dada la imparable evolución de las tecnologías, se deben crear métodos o formas de verificar o autenticar el origen de los datos y dar credibilidad de la fuente.

2.2.1. HDFS

Por ello, The Apache Software Foundation[10] desarrolló un Sistema de Ficheros, HDFS[24][1], basado en la idea de Google de su propio Sistema de Ficheros. HDFS es un Sistema de Ficheros Distribuido pensado para el almacenamiento de ficheros de gran tamaño permitiendo su acceso por múltiples clientes a la vez.

En la figura 2.1 se muestra la arquitectura utilizada por HDFS. HDFS [1] se caracteriza por tener un Datanode donde se almacenan los bloques de datos de los ficheros que los usuarios decidan introducir en el Sistema, no obstante, estos bloques de datos tienen diferentes copias de ellos para tener tolerancia a fallos en caso de que ocurra algún problema. También posee un Namenode que es donde almacena los metadatos de los ficheros que se almacenan en el Sistema como puede ser su nombre o sus permisos de lectura y escritura. Como se puede comprobar, su comportamiento con respecto a los ficheros es similiar al Sistema de Ficheros usado por UNIX.

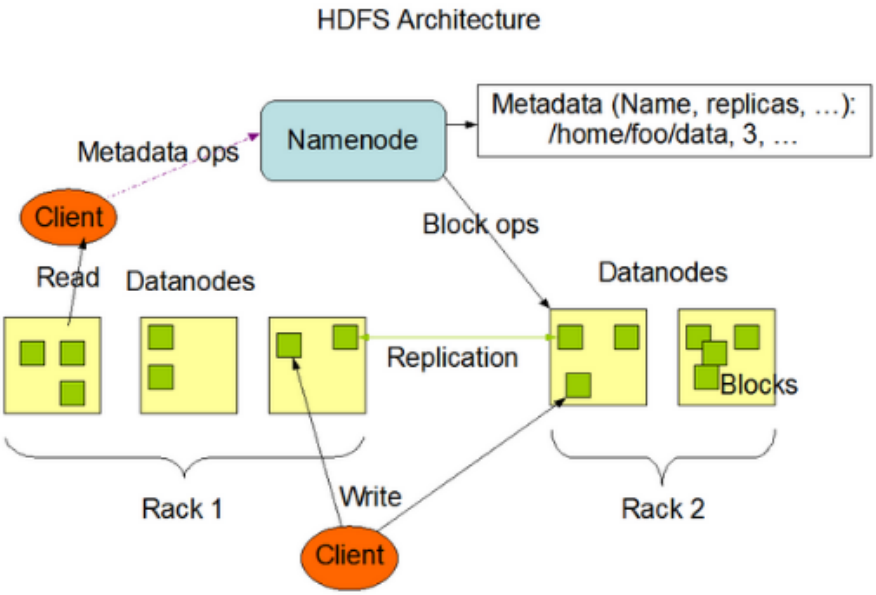


Fig. 2.1. Arquitectura del Sistema de Ficheros HDFS

A pesar de que HDFS [1][24]ofrece la capacidad de realizar unas escrituras a gran velocidad, no permite que varios clientes puedan escribir en un mismo fichero a la vez, pero sí permite lecturas paralelas, es decir, HDFS se basa en el modelo de un único escritor y varios lectores. Lo que hace este Sistema de Ficheros es proteger el fichero en su totalidad mientras un único cliente está escribiéndolo, de tal forma que no pueda haber escrituras simultáneas. Sin embargo, este modelo permite que, una vez se ha terminado de escribir, varios clientes pueden leerlo. Se puede comprobar que HDFS da prioridad a la consistencia de la información en este caso.[25]

2.2.2. Map-Reduce

Después de explicar de forma resumida el Sistema de Ficheros base que da motivo a la realización de este proyecto, se va a explicar esta técnica de procesado de datos, muy utilizado para la creación de conjuntos de datos en un periodo de tiempo que permita su uso de manera rápida y eficiente. Este programa es muy conocido y utilizado actualmente porque permite ejecutar de forma paralela el mismo programa en decenas o cientos de máquinas a la vez.

Esta técnica, llamada Map-Reduce[26], se caracteriza por estar formada por dos funciones, llamadas Map y Reduce, ambas desarrolladas por el propio usuario, pero que, una

vez implementadas, ofrecen la capacidad de ser usada para muchos otros programas. En el caso del que se trata en esta memoria de tratamiento de ficheros se puede contar el número de accesos a un fichero, o la cantidad de veces que un usuario ha escrito una palabra, entre muchas otros programas.

Map

Para explicar esta técnica se va a utilizar el ejemplo de contar palabras mencionado al principio de este subapartado. La función Map va a recibir un gran conjunto de datos y va a generar una nueva tupla de datos <Clave,Valor>[26]. Más concretamente, en la Figura 2.2 lo que recibe Map será un par <Clave, Valor>, siendo la clave el documento y el valor será la palabra que se va a tener en cuenta para obtener el número de ocurrencias del documento. Por cada palabra que coincida con el valor deseado, se emitirá un par de claves indicando que ha encontrado una ocurrencia, de tal forma que:

```
map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
      EmitIntermediate(w, "1");
```

Fig. 2.2. Ejemplo de Función Map

Reduce

Por otro lado, la función Reduce recibe estos pares <Clave, Valor>, siendo en este caso el valor igual 1 (<Clave,1>) por cada ocurrencia del fichero de la palabra que se desea buscar, lo que hará es ir sumando de uno en uno al resultado global el número total de ocurrencias existentes. En la Figura 2.3 se puede ver un pseudocódigo posible para la función Reduce:

```
reduce(String key, Iterator values):
   // key: a word
   // values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);
   Emit(AsString(result));
```

Fig. 2.3. Ejemplo de Función Reduce

2.3 Apache Cassandra

Tras explicar el funcionamiento de Map-Reduce[26], es necesario detallar el funcionamiento del que será la base del presente proyecto que se está explicando. Apache Cassandra[11] es una Base de Datos Distribuida, que, además, es descentralizada, tolerante a fallos, escalable linealmente y está orientado al almacenamiento de datos por columnas[27][28]. A continuación, se van a explicar las diferentes características previamente enumeradas:

- Distribuida: Esto permite que la Base de Datos se encuentre desplegada en un cluster, esto es, un conjunto de ordenadores o componentes unidos entre sí por una red que se comportan como una única computadora, que puede estar geográficamente en diferentes lugares.
- Descentralizada: Esto quiere decir que no existe un nodo maestro que controle las comunicaciones, además de que no habría cuello de botella con el paso de información.
- Tolerante a fallos: Si algún nodo del cluster falla, se puede sustituir con facilidad y la información, al estar replicada, puede restablecerse de nuevo en el nodo que ha tenido el problema.
- Escalable linealmente: Esto quiere decir que a mayor número de nodos que se incorporen al cluster, las peticiones que se ofrezcan por segundo por nodo no se reducirán, además de que el rendimiento aumentará con cada nodo extra.
- Orientado a columnas: En este tipo de Base de Datos la información se almacena en columnas en lugar de filas, teniendo como objetivo escribir y leer datos de forma

eficiente desde y hacia el disco duro. Por ejemplo, los valores de, la primera columna, están físicamente juntos, seguidos de los valores de la segunda columna, etc. En la figura 2.4 se puede comprobar el funcionamiento de esta idea.

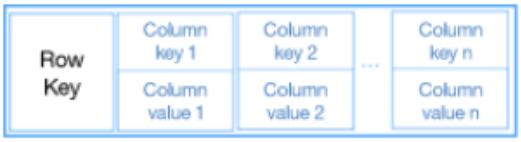


Fig. 2.4. Estructura de partición de datos de Cassandra

Partiendo de la figura 2.5 donde se puede ver la arquitectura de la Base de Datos, se puede comprobar que consiste en un anillo en el que los nodos que conforman el cluster se ponen de acuerdo para repartirse unos "tokens" de forma equitativa para repartir la carga de datos entre ellos. Estos tokens se obtienen de un particionador (Partitioner) que es una función Hash que determina el rango posible de claves de fila, que son los identificadores de las filas.

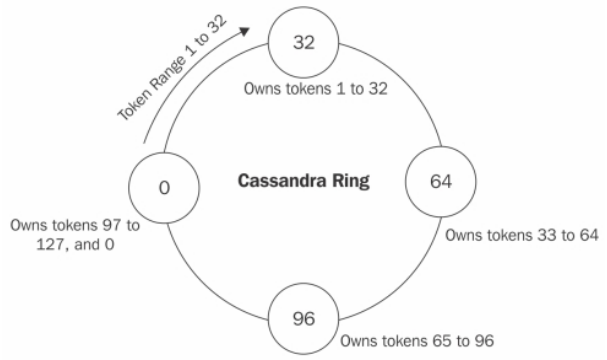


Fig. 2.5. Distribución de tokens en un anillo de un cluster de Cassandra

Para poder explicar el funcionamiento de las escrituras y lecturas en Cassandra[28] es necesario describir antes algunos de los componentes más importantes de los que hace uso Cassandra que aparecen en la figura 2.6:

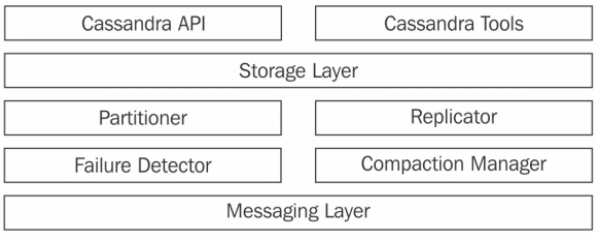


Fig. 2.6. Componentes de Cassandra

- Storage Layer: Esta capa es la responsable de manejar todas las peticiones que los clientes hagan a la Base de Datos
- Messaging Layer: Esta capa tiene como objetivo establecer comunicación entre los diferentes nodos que conforman el cluster. Cassandra usa el "gossip", para poder comunicarse entre nodos basándose en la idea de emitir información como si se tratase de un rumor, que se transfiere a toda la población permitiendo a los nodos hacerse un mapa global del sistema.
- Failure Detector: Este componente es uno de los más fundamentales en un sistema distribuido. La idea de este componente yace en detectar un fallo de comunicación y actuar en base al estado del nodo que ha fallado. Para saber esto, se utiliza el método de comunicación antes descrito para saber el estado de los nodos.
- Partitioner: Cuando un usuario inserta información, Cassandra asigna cada fila una clave de fila anteriormente explicada y, en base a esto, determina qué nodo es el encargado de mantener esa información.
- Replicator: Para evitar la pérdida de datos que estén almacenados en un nodo, Cassandra ofrece la posibilidad de replicar esa información a otro nodo del cluster, ofreciendo tolerancia a fallos.

A la hora de escribir en la Base de Datos, los clientes pueden conectarse cualquier nodo que forme parte del cluster de Cassandra[28] y enviar datos para escribir. Cuando uno de los nodos recibe esta petición del cliente, se activa el servicio que se encuentra en el componente Storage Layer anteriormente descrito. Este componente obtendrá todos los nodos que son responsables de guardar la información que se va a escribir. Una vez que el nodo coordinador de la escritura sepa los nodos donde se puede escribir, les enviará

un mensaje a la espera de recibir una respuesta de que puede enviarlo. No obstante, no espera a todos ellos, sino simplemente esperará cuando reciba el mínimo de respuestas que coincida con el factor de replicación de la tabla donde se quiera almacenar los datos del usuario. A continuación, en la figura 2.7, se muestra el comportamiento de la escritura en Cassandra:

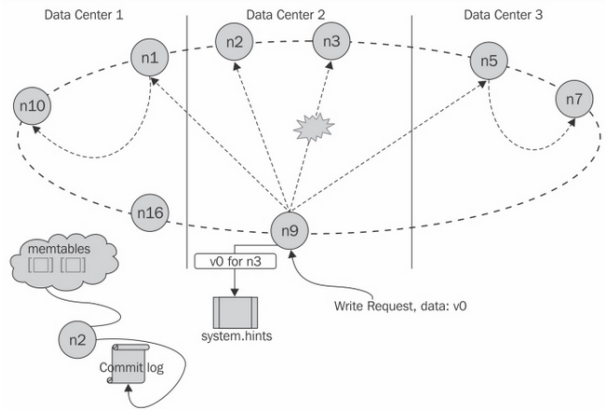


Fig. 2.7. Escritura en Cassandra

El funcionamiento de la lectura es similar a la escritura como se ve en la figura 2.8, cuando un usuario se conecta a cualquier nodo para leer algún dato en específico, el nodo pide una lista de todos los nodos que contienen esa información. Una vez que lo tiene gracias a la estrategia de replicación que use Cassandra, el nodo coordinador ordenará por proximidad los nodos con respecto a él. Este orden se define en base a una de las siguientes estrategias determinadas por la función del "soplón" (Snitch)[28]:

- SimpleSnitch: El nodo más cercano será aquel que se encuentre a continuación en el anillo siguiendo el sentido horario
- AbstractNetworkTopologySnitch: Partiendo del hecho de que hay que distinguir un rack de un datacenter, mientras que un rack es un conjunto formado por aquellos nodos que están mas cerca, un datacenter es un conjunto de racks que no tienen porque ser próximos entre ellos. Teniendo esto en cuenta, esta estrategia funciona de la siguiente forma: Para un nodo, el siguiente más cercano será aquel que se encuentre en su mismo rack, seguido del nodo que se encuentre en el mismo datacenter pero en distinto rack y, por último, será aquel nodo que esté en el datacenter más cercano.

- DynamicSnitch: Esta última estrategia tiene en cuenta el rendimiento de cada nodo, por lo que un nodo que responde de manera más rápida se tendrá en cuenta antes indepdendientemente de la distancia a la que se encuentre de otro más cercano al nodo coordinador.

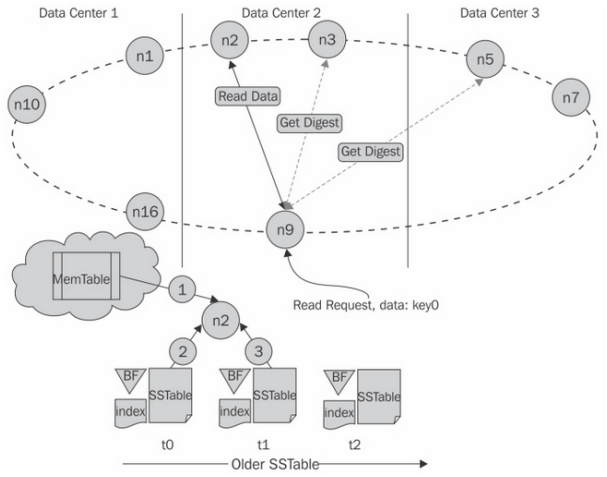


Fig. 2.8. Lectura en Cassandra

Como se puede ver en las figuras 2.7 y 2.8, aparecen varios elementos que no han sido mencionados hasta ahora. Estos elementos son el MemTable, el SSTable y el CommitLog.

- MemTable es una estructura que se almacena en memoria con forma de tabla hash que almacena la información de las columnas. Esta memoria escribe en orden hasta alcanzar el máximo posible de información a escribir, para luego purgarlo.

- SSTable es la versión en disco de las MemTables, de tal forma que cuando se purgan de memoria estas últimas, las SSTables se persisten en el disco duro

- El CommitLog recibe todas las escrituras que se han hecho en un nodo del cluster, y estas escrituras se mantienen de manera permanente incluso si el nodo falla.

2.3.1. NoSQL

Junto con lo que se ha dicho sobre el funcionamiento de Apache Cassandra, hay que añadir el hecho de que es una Base de Datos no relacional, es decir, es una Base de Datos NoSQL[27] [12]. El término NoSQL quiere decir que no solo usa SQL [29] ("Not

Only SQL"), sino que además combina este tipo de Base de Datos con aquellas que son relacionales.

Este tipo de BBDD son relativamente recientes y, además, está siendo utilizado por grandes compañías como pueden ser Facebook, Twitter o Amazon, entre otros.

Las BBDD no relacionales sacrifican una norma muy usada por las relacionales, que son las transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) en favor de otro tipo de transacciones que yacen en el Teorema CAP[30] (Consistencia, Disponibilidad y Tolerancia al particionado)[28]. Cassandra, por ejemplo, se centra en la Disponibilidad y en la Tolerancia al particionado. En la figura 2.9 se muestra una comparativa de Cassandra con otras BBDD no relacionales que siguen el Teorema CAP.

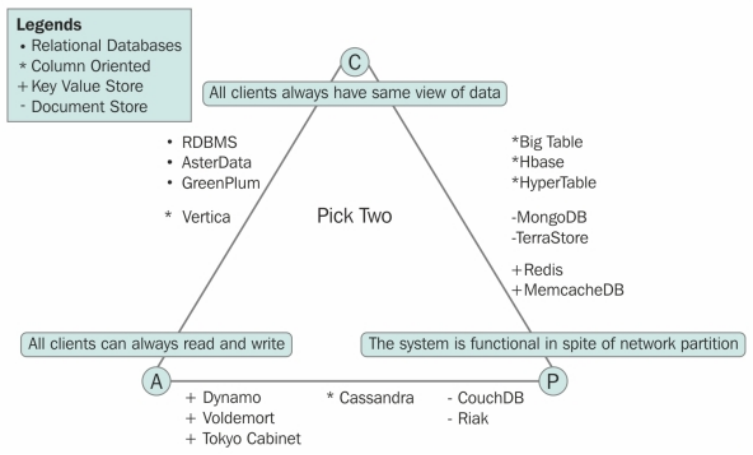


Fig. 2.9. Comparativa CAP entre BBDD

2.3.2. Cassandra Query Language

Una vez se ha explicado el funcionamiento de este tipo de bases de datos, conviene detallar exactamente cuál es el lenguaje que usa Apache Cassandra para que el usuario pueda realizar operaciones de lectura o escritura.

Cassandra usa CQL[31][32], lenguaje de consultas propio de esta BBDD que se parece al lenguaje SQL [29] tradicional pero con ciertos retoques o cambios que permiten que Cassandra sea un poco más flexible. Además, para que el usuario pueda interactuar con mayor facilidad con Cassandra, éste ofrece un intérprete de comandos llamado cqlsh, necesario para poder crear las keyspaces que serán los espacios de memoria donde se podrán crear las tablas y, por ende, insertar y modificar la información que esté dentro de

ellas.

Estas diferencias entre ambos lenguajes se resumen en los siguientes puntos a considerar si se quiere usar una Base de Datos como Cassandra u otra que tenga SQL :

- Si se quiere hacer alguna búsqueda de algún dato, que no sea la clave primaria (es decir, aquella clave que es única por fila), se deberá crear un índice en aquella columna por la que se desea buscar. Esta creación de índices no es obligatoria si se quiere utilizar SQL[29], pero en CQL[31] se debe hacer.
- CQL solo ofrece la posibilidad de utilizar el operador lógico AND, por lo que se debe tener muy en cuenta a la hora de modelar la información en Cassandra.
- Cassandra maneja de forma diferente las fechas, permitiendo solamente el operador de igualdad para las marcas de tiempo o "timestamps", mientras que SQL permite utilizar otros operadores además de expresiones del tipo date >TO_TIMESTAMP ('2015-01-05','YYYY-MM-DD').
- CQL no soporta agrupaciones (GROUP BY) ni tampoco JOINS, ni disparadores, elementos muy utilizados en SQL .

No obstante, Cassandra ofrece otras características que permiten que no se vea afectada por la falta de los elementos listados anteriormente.

Este intérprete de comandos forma parte de uno de los componentes de Cassandra que aparece en la figura 2.6 anteriormente explicada, más concretamente, forma parte de Cassandra Tools, junto con otras herramientas para poder manejar el cluster como puede ser Nodetool, entre otros.

Por último, cabe destacar los diferentes tipos de datos que soporta Cassandra a la hora de manejar los datos que los clientes deseen introducir. En la siguiente figura se muestra los diferentes datos que permite[32]:

Internal Type	CQL Name	Description
BytesType	blob	Arbitrary hexadecimal bytes (no validation)
AsciiType	ascii	US-ASCII character string
UTF8Type	text, varchar	UTF-8 encoded string
IntegerType	varint	Arbitrary-precision integer
LongType	int, bigint	8-byte long
UUIDType	uuid	Type 1 or type 4 UUID
DateType	timestamp	Date plus time, encoded as 8 bytes since epoch
BooleanType	boolean	true or false
FloatType	float	4-byte floating point
DoubleType	double	8-byte floating point
DecimalType	decimal	Variable-precision decimal
CounterColumnType	counter	Distributed counter value (8-byte long)

Fig. 2.10. Tipos de Datos aceptados por Cassandra

2.4 POSIX

POSIX [33] es un estándar o norma que define la interfaz estándar del Sistema Operativo y el entorno, ofreciendo también un intérprete de comandos o "shell". Este estándar fue desarrollado y registrado por el IEEE y de los componentes principales de los que consta, cabe destacar que ofrece convenciones de utilidades, definiciones de cabeceras o "headers" para el lenguaje de programación C, definición de funciones y subrutinas y, además, ofrece definiciones para el desarrollo de un intérprete de comandos y utilidades para aplicaciones que se incluyen en esta "shell".

Este estándar, trasladado al proyecto que ocupa esta memoria, utilizará esas definiciones de las funciones o subrutinas de las cabeceras que presenta el estándar para poder transformarlas en funciones similares y permitir crear un Sistema de Ficheros capaz de crear y manipular directorios y ficheros regulares.

Además, el propio intérprete de comandos del que se menciona en este apartado es utilizado en el Sistema Operativo UNIX [34] y proporciona la información necesario de las funciones que van a ser necesarias y aquellas en las que más interés se tenga para el correcto funcionamiento del presente proyecto.

2.5 MPI

MPI [17], como se ha dicho en la Introducción de esta memoria, es una biblioteca de paso de mensajes diseñado para que los programas utilicen lo máximo posible la potencia que ofrecen los procesadores. Esta interfaz se usa, principalmente, en el campo de la ingeniería puesto que permite utilizar técnicas de programación concurrente con varios procesos a través de la sincronización entre ellos y de la exclusión mutua, al igual que funcionan los semáforos o los cerrojos.

Las aplicaciones que utilizan esta interfaz siguen un modelo de programación paralela en el que los mensajes son enviados desde distintos procesos con espacios de memoria diferentes mediante operaciones que realizan conjuntamente. Entre los distintos mensajes que se pueden enviar entre ellos destacan: el envío de un mensaje de un proceso a otro proceso, el envío de un mensaje de un único proceso al resto de procesos que estén ejecutando, conocido como envío Broadcast; el envío de mensajes de diferentes procesos a un único proceso, conocido como envío Gather; y, por último, el envío Multicast, el cuál envía diferentes mensajes de varios procesos a más de un proceso.

Dentro de MPI también existe una definición de una interfaz que permite acceder a diferentes Sistemas de Ficheros de manera compartida, conocida como interfaz MPI-IO [35]. Esta interfaz, como se verá en la sección 2.5.1 establece que, para poder funcionar correctamente, designa un identificador a cada uno de los procesos que forman parte de la aplicación, desde 0 hasta n-1, y también, para que estos procesos puedan comunicarse entre ellos, establece una serie de comunicadores que indican cuáles de los procesos deben formar parte de las operaciones que tengan que realizar conjuntamente.

2.5.1. MPI-IO

Para este proyecto lo que se pretende conseguir es integrar, de forma parcial, las funciones que se desarrollen para acceder a Apache Cassandra en MPI, de tal forma que se puedan utilizar diversas funciones que ofrece MPI para acceder a la Base de Datos y, además, se pueda realizar utilizando varios procesos gracias a MPI-IO [35].

El objetivo de utilizar esta biblioteca que ofrece MPI consiste en centrar las bases para una implementación completa, puesto que, en primera instacia, se pretende comprobar

la viabilidad del sistema a desarrollar en este entorno. Todo el trabajo de integración a realizar de este nuevo Sistema de Ficheros dentro de MPICH [36].

Por último, se va explicar las implementaciones que ofrece MPI-IO que se usan actualmente: Romio y ADIO, y que es necesaria para la integración de la interfaz que se desarrolle en el proyecto.

Romio

Romio [37] es una implementación de alto rendimiento y portable de MPI-IO que implementa un par de técnicas cuyo objetivo es optimizar la entrada y la salida, lo que permite mejorar el rendimiento de las aplicaciones que usen Romio. En primer lugar, Romio está optimizado para acceder a partes que no están contiguas en un fichero gracias a la técnica del *data sieving*, y, en segundo lugar, Romio utiliza un conjunto de agregadores que accederán al fichero y obtendrán la información necesaria que luego será dividida y distribuida entre los procesos que formen parte de una operación colectiva, que se conoce con el nombre de *two phase I/O*[38].

ADIO

ADIO [39] es una estrategia que permite implementar cualquier interfaz de entrada-salida paralela a nivel de usuario, en este caso, interfaces para sistemas de ficheros [6]. Más concretamente, consiste en un conjunto de funciones que permiten la manipulación de ficheros de forma paralela, tanto escritura como lectura, pero se deben adaptar a cada uno de los diferentes sistemas que se vayan a usar.

Capítulo 3

Análisis

En este capítulo se procederá a examinar el proyecto que se ha llevado a cabo, más concretamente, se desarrollará una descripción del proyecto realizado (ver sección 3.1). Después, se explicarán las diferentes opciones que podían ser usadas para completar el trabajo (ver sección 3.2) y se realizará un análisis de los requisitos (ver sección 3.3) de software necesarios para poder cumplir con mayor detalle y precisión las necesidades que deben satisfacerse para que los usuarios puedan usarlo. Por último, se explicará el marco regulatorio que seguirá el presente trabajo (ver sección 3.4)

3.1 Descripción del proyecto realizado

Como se mencionó en el apartado de objetivos del capítulo 1, entre los objetivos de este proyecto destacan principalmente dos, que son el diseño e implementación de una interfaz basada en POSIX[33] sobre la Base de Datos Apache Cassandra y comparar los resultados obtenidos en relación con tasas de transferencia con el sistema creado con la tasa de transferencia que es capaz de manejar el Sistema de Ficheros HDFS [1].

Para poder empezar a determinar cómo empezar el desarrollo del Sistema de Ficheros que se quiere hacer en Cassandra, es necesario tener una idea de cómo POSIX, estándar del que se tiene que basar el desarrollo del proyecto, maneja y trata los ficheros regulares y los directorios. Eligiendo como ejemplo un Sistema Operativo UNIX[34] , POSIX ofrece varias funciones, en el lenguaje de programación C tal y como se explicó en el apartado de Estado del Arte, de las cuales se tendrán en cuenta un número reducido de ellas, que son las más importantes a la hora de tratar con este tipo de elementos. En la siguiente

figura se muestra un esquema simplificado de las funciones más importantes:

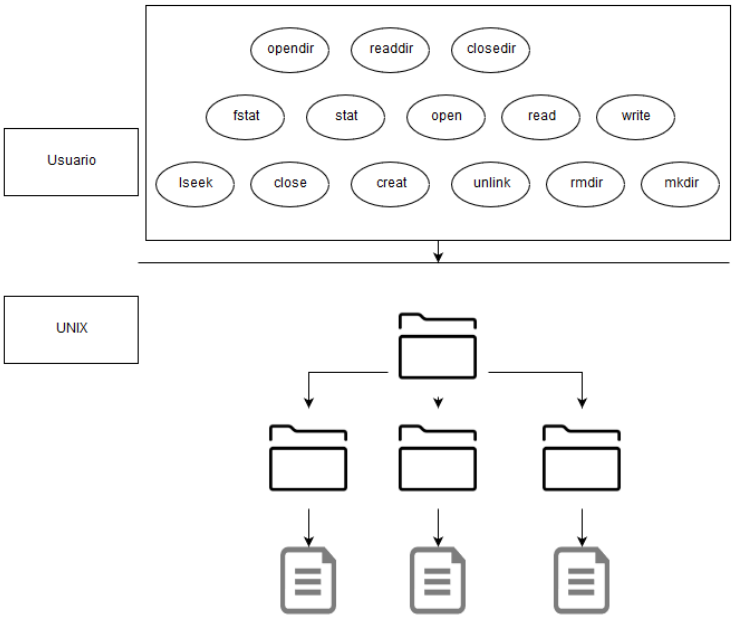


Fig. 3.1. Funciones de manejo de directorios y ficheros POSIX

Además, también hay que tener en cuenta el estado inicial en el que se encuentra la Base de Datos y, por tanto, también hay que determinar cómo se puede usar Cassandra sin ningún tipo de apoyo que permita poder comunicarse haciendo uso de algún lenguaje de programación. Es por ello que, inicialmente, la única forma de poder acceder a Cassandra es de la siguiente forma:

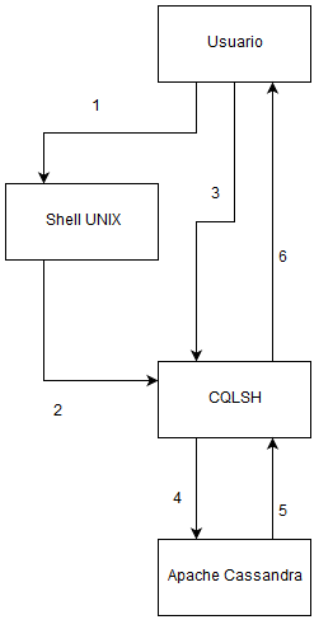


Fig. 3.2. Estado inicial de Cassandra

- 1.- Se abre un intérprete de comandos en UNIX y se introduce el comando para poder acceder a la shell que ofrece Apache Cassandra, introduciendo la dirección IP en la que se encuentra arrancado uno de los nodos que conformen el cluster.
- 2.- El intérprete de comandos se conecta al componente que ofrece Cassandra para ejecutar la shell cqlsh y a partir de este momento toda la interacción se centra entre el usuario y la shell de Cassandra.
- 3.- El usuario introduce una consulta de cualquier tipo en la shell.
- 4.- El intérprete de comandos de Cassandra evalúa sintácticamente la consulta y envía la petición a la Base de Datos.
- 5.- Apache Cassandra devuelve el resultado o un error en caso de no haber encontrado lo que el usuario pedía.
- 6.- La shell recibe el resultado y muestra al usuario lo que pedía o un error en su defecto.

Por lo tanto, lo que se debe buscar es algún driver o conector que permita conectarse a la Base de Datos directamente, utilizando un lenguaje de programación determinado y

realizar llamadas a Cassandra para poder realizar todas las peticiones necesarias y que, además, sigan el funcionamiento que POSIX a la hora de crear, modificar, eliminar ficheros y directorios.

3.2 Elección de la solución

En esta sección lo que se quiere conseguir es seleccionar la opción más viable para desarrollar el proyecto de entre una amplia variedad de lenguajes de programación, además de que tengan soporte para utilizar un driver que permita conectarse a Cassandra y poder realizar las operaciones deseadas. Por ello, en la siguiente tabla, se muestra una comparativa de los diferentes lenguajes de programación que se pueden tener en cuenta a la hora de realizar el proyecto:

TABLA 3.1. COMPARATIVA DE LENGUAJES DE PROGRAMACIÓN

Lenguaje de Programación	C	C++	C#	Java	Python	Nodejs	PHP
Tiene soporte para driver	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Soporta última versión Cassandra	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Ofrece similitudes con estándar POSIX	Sí	Sí	No	No	No	No	No

Para la tabla 3.1 anteriormente mostrada, hay que tener en cuenta las siguientes características:

- Tiene soporte para driver: Como su propio nombre describe, indica si el lenguaje de programación tiene la posibilidad de utilizar algún driver que le permita conectarse con Cassandra.
- Soporta última versión Cassandra: Otro tema a tener en cuenta es que si el lenguaje tiene la posibilidad de usar algún driver, también este ofrece la posibilidad de conectarse a la última versión existente de la Base de Datos.

- Ofrece similitudes con estándar POSIX: Esto se debe tomar en consideración puesto que POSIX ofrece cabeceras para el lenguaje de programación C y, por lo tanto, es necesario saber si existe algún otro lenguaje que presente alguna similitud de la que se pueda partir a la hora de desarrollar el proyecto

Partiendo de la tabla 3.1, se ha decidido utilizar como lenguaje de programación C, que tiene la posibilidad de usar un driver para conectarse a Cassandra y, además, este lenguaje de programación es el que se usa en el estándar POSIX para poder manejar ficheros y directorios, por lo que resulta más sencillo trasladar a este proyecto su funcionamiento. Por otro lado, este lenguaje forma parte de los requisitos no funcionales que posteriormente se definirán establecido por el tutor del presente proyecto.

3.3 Requisitos

En esta sección se describirán todos los requisitos que han sido necesarios y se han tenido en cuenta para la realización de este proyecto y más concretamente para el diseño e implementación de la interfaz para acceder a Cassandra mediante el driver de C que proporciona la empresa Datastax[40] , empresa que, además, tiene los drivers más actualizados en varios lenguajes de programación en comparación con los demás drivers que aparecen en la página web de Apache Cassandra.

Para poder realizar de forma correcta el análisis de requisitos de Software, se han seguido las especificaciones de buenas prácticas que ofrece IEEE[41]. Antes de empezar, cabe destacar que, según el IEEE, un requisito es aquella condición o capacidad pedida por un usuario para resolver un problema o alcanzar un objetivo, o también una condición o capacidad que debe tener un sistema para satisfacer un contrato o cualquier otro documento que se haya impuesto formalmente[42].

Según el IEEE, una buena especificación de requisitos de software debería caracterirzarse por ser:

- Correcta: No tiene que haber errores.
- Consistente: Todos los requisitos son coherentes entre ellos.
- Completa: Todos los requisitos tienen que estar en la especificación y tener todas

las referencias.

- Verificable: Existe algún método finito que pueda comprobar cada requisito.
- Trazable: Existe una historia clara de cada requisito.
- Modificable: Los requisitos se pueden cambiar fácilmente.
- Clara y no ambigua: Los requisitos no tienen más de una interpretación.
- Priorizable: Los requisitos pueden organizarse por orden de importancia clasificándolos por esenciales u opcionales.

Para la especificación de requisitos, se van a describir los Requisitos de Software, que proporcionarán una información más detallada y formal sobre el funcionamiento del sistema que el usuario desea.

3.3.1. Requisitos de Software

Una vez que se han explicado los casos de uso necesarios para este sistema, es necesario distinguir entre los dos tipos de requisitos software que se van a tratar en esta sección, después de los requisitos de usuario ya explicados anteriormente:

- **Requisitos funcionales**: Son características o funcionalidades que el sistema debe tener.
- **Requisitos no funcionales**: Son restricciones que se imponen al sistema que deben cumplir. No obstante, dentro de este tipo de requisitos, se pueden subdividir en un amplio abanico de tipos de requisitos no funcionales[42]:
  - **Requisitos de Fiabilidad**: Describen la medida en que un sistema debe funcionar correctamente, es decir, el número de fallos permitidos.
  - **Requisitos de Rendimiento**: Se definen los recursos que se pueden usar para el funcionamiento del sistema, ya sea uso de CPU o memoria.
  - **Requisitos de Interfaz**: Indica el modo con el que el sistema se debe comunicar con otros sistemas o componentes.

- **Restricciones**: Determina las limitaciones del sistema
- **Requisitos de Seguridad**: Especifican la forma de asegurar el sistema contra ataques a la confidencialidad, integridad o disponibilidad.
- **Requisitos de Manejo de Errores**: Define cómo el sistema debe actuar frente a fallos
- **Requisitos de Rendimiento**: Indica las tasas de transferencia o la velocidad del proceso de ejecución del sistema.

Tras haber descrito las características de los distintos tipos de requisitos que se pueden encontrar en el análisis de un sistema, se procede a definir los requisitos que se han tenido en cuenta para el presente proyecto, no sin antes definir una tabla de ejemplo que incluya los campos que se han utilizado:

TABLA 3.2. EJEMPLO DE TABLA DE DEFINICIÓN DE REQUISITOS

Identificador	Identificador del requisito formado por una letra que indica si es un requisito funcional (RF) o no funcional (RNF), seguido de un número de tres dígitos.
Título	Nombre del requisito.
Prioridad	Campo que indica la importancia del requisito a la hora de desarrollarlo. Puede ser Prioridad alta, Prioridad media, Prioridad baja.
Descripción	Explicación más exhaustiva del requisito.
Fuente	Origen del que proviene la creación del requisito.
Estabilidad	Campo que indica si el requisito cambiará con el paso del tiempo. En caso de que cambie tendrá una estabilidad baja. En caso contrario, tendrá una estabilidad alta.
Pruebas de Verificación	Campo en el que se introducirán casos de comprobación de la correcta ejecución del requisito.

Requisitos Funcionales

TABLA 3.3. REQUISITO FUNCIONAL RF-001

Identificador	RF-001
Título	Conexión con Apache Cassandra
Prioridad	Alta
Descripción	El sistema deberá poder conectarse a cualquier nodo del cluster que forme parte de la Base de Datos.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Seleccionar una dirección IP que forme parte del cluster para comprobar que se conecta. - Seleccionar una dirección IP que no forme parte del cluster para comprobar que no se conecta.

TABLA 3.4. REQUISITO FUNCIONAL RF-002

Identificador	RF-002
Título	Desconexión de Apache Cassandra
Prioridad	Media
Descripción	El sistema podrá desconectarse de la Base de Datos cuando termine de ejecutarse el programa del usuario.
Fuente	Tutor
Estabilidad	Baja
Pruebas de Verificación	- Ejecutar un programa sin utilizar la Conexión a la Base de Datos para determinar si anteriormente ya estaba conectado.

TABLA 3.5. REQUISITO FUNCIONAL RF-003

Identificador	RF-003
Título	Función c_creat
Prioridad	Alta
Descripción	Desarrollar la función c_creat para crear un fichero que antes no existía en la Base de Datos, recibiendo como entrada el directorio absoluto de creación con el nombre del fichero, seguido de los permisos del fichero.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Crear un fichero que no existía antes en la Base de Datos para comprobar que se crea correctamente. - Crear un fichero que ya existe en la Base de Datos en el mismo directorio. - No insertar el nombre del fichero para comprobar que no se ha introducido de manera correcta los campos. - No insertar el path absoluto de creación del fichero para recibir un error.

TABLA 3.6. REQUISITO FUNCIONAL RF-004

Identificador	RF-004
Título	Función c_open
Prioridad	Alta
Descripción	Desarrollar la función c_open que permita abrir un fichero existente o que, en caso de que no exista, permita crear el fichero en cuestión y, posteriormente, abrirlo. Esta función recibirá el path absoluto del fichero y los flags de manejo del fichero y devuelve el descriptor de fichero asociado.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Introducir un fichero que ya esté creado en la Base de Datos para comprobar que se obtiene su descriptor de fichero correctamente. - Abrir un fichero que no existe en la Base de Datos sin el flag O_CREAT, en cuyo caso dará error por no existir. - Abrir un fichero que no existe en la Base de Datos insertando el flag O_CREAT, en cuyo caso creará el fichero y devolverá el descriptor de fichero correspondiente. - Abrir un fichero que existe en la Base de Datos insertando el flag O_TRUNC, en cuyo caso vaciará el fichero en caso de que tuviese alguna información previa. - Abrir un fichero que existe en la Base de Datos insertando el flag O_APPEND, en cuyo caso abrirá correctamente el fichero y pondrá el puntero de lectura y escritura al final del fichero. Si no se pone este flag, el puntero se pone por defecto en la posición inicial del fichero. - Abrir un fichero en un directorio que no existe, cuyo resultado será un error por no existir el directorio. - No introducir ningún path dará como resultado un error por no ser posible introducirlo.

TABLA 3.7. REQUISITO FUNCIONAL RF-005

Identificador	RF-005
Título	Función c_close
Prioridad	Alta
Descripción	Desarrollar la función c_close que permita cerrar el fichero que el usuario ha abierto previamente introduciendo el descriptor de fichero asociado.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Introducir un descriptor de fichero que el proceso del cliente tenga abierto cerrará correctamente el fichero. - Introducir un descriptor de fichero que el proceso del cliente no tenga abierto dará lugar a un error.

TABLA 3.8. REQUISITO FUNCIONAL RF-006

Identificador	RF-006
Título	Función c_fstat
Prioridad	Alta
Descripción	Desarrollar la función c_fstat que permite obtener los metadatos de los ficheros regulares almacenados en la Base de Datos a partir del path absoluto de dicho fichero junto con un puntero a una estructura donde almacenar los metadatos encontrados.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Insertar un descriptor de un fichero regular que esté abierto por el cliente dará lugar a la obtención de manera correcta de los metadatos del fichero indicado. - Introducir un descriptor de fichero de un fichero que no esté abierto por el cliente dará lugar a un error.

TABLA 3.9. REQUISITO FUNCIONAL RF-007

Identificador	RF-007
Título	Función c_lseek
Prioridad	Alta
Descripción	Desarrollar la función c_lseek cuya función consiste en mover el puntero de posición de un fichero abierto que recibe con su descriptor de fichero, junto con el offset y el whence (este último indica si el puntero se pone al final del fichero, al principio o en la actual posición en la que se encuentre).
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Introducir un descriptor de fichero que no se encuentra entre los ficheros abiertos por el cliente dará lugar a un error. - Introducir un descriptor de fichero que se encuentra abierto por el proceso cliente junto con un whence que no sea SEEK_SET, SEEK_CUR o SEEK_END dará error. - Introducir un descriptor de fichero que se encuentra abierto por el proceso cliente junto con un whence que sea SEEK_SET y un offset cuya suma de una posición positiva permitirá cambiar el puntero a la posición deseada. - Introducir un descriptor de fichero que se encuentra abierto por el proceso cliente junto con un whence que sea SEEK_CUR y un offset cuya suma de una posición positiva permitirá cambiar el puntero a la posición deseada. - Introducir un descriptor de fichero que se encuentra abierto por el proceso cliente junto con un whence que sea SEEK_END y un offset cuya suma de una posición positiva permitirá cambiar el puntero a la posición deseada. - Introducir un descriptor de fichero que se encuentra abierto por el proceso cliente junto con un whence que sea SEEK_SET, SEEK_CUR o SEEK_END y un offset cuya suma de una posición negativa dará lugar a error.

TABLA 3.10. REQUISITO FUNCIONAL RF-008

Identificador	RF-008
Título	Función c_stat
Prioridad	Alta
Descripción	Desarrollar la función c_stat que permite obtener los metadatos de los ficheros regulares almacenados en la Base de Datos a partir del path absoluto de dicho fichero junto con un puntero a una estructura donde almacenar los metadatos encontrados.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Insertar un path absoluto de un fichero regular que existe en la Base de Datos dará lugar a la obtención de manera correcta de los metadatos del fichero indicado. - Introducir el path absoluto de un directorio que no existe dará lugar a un error. - Introducir el path absoluto de un directorio que existe permitirá obtener los metadatos del directorio a pesar de que esta función está centrado en los ficheros regulares.

TABLA 3.11. REQUISITO FUNCIONAL RF-009

Identificador	RF-009
Título	Función c_write
Prioridad	Alta
Descripción	Desarrollar la función c_write que permite escribir en la Base de Datos el fichero regular deseado asociado al descriptor de fichero que recibe, junto con el buffer que se quiere escribir más su tamaño en bytes.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Insertar un descriptor de fichero que no se encuentre abierto dará lugar a un error. - Si el descriptor de fichero que recibe la función es correcto y el tamaño del buffer que se quiere introducir es un número mayor o igual que 0, el buffer se escribirá correctamente en el fichero desde la posición en la que estuviera el puntero en el momento de la llamada a la función.

TABLA 3.12. REQUISITO FUNCIONAL RF-010

Identificador	RF-010
Título	Función c_read
Prioridad	Alta
Descripción	Desarrollar la función c_read que permite leer de la Base de Datos el fichero regular deseado asociado al descriptor de fichero que recibe, junto con el buffer que se quiere leer más su tamaño en bytes.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Insertar un descriptor de fichero que no se encuentre abierto dará lugar a un error. - Si el descriptor de fichero que recibe la función es correcto y el tamaño del buffer que se quiere leer es un número mayor o igual que 0, el tamaño del fichero se leerá correctamente desde la posición en la que estuviera el puntero en el momento de la llamada a la función y se escribirá en el buffer.

TABLA 3.13. REQUISITO FUNCIONAL RF-011

Identificador	RF-011
Título	Función c_opendir
Prioridad	Media
Descripción	Desarollar la función c_opendir que permite abrir un directorio y obtener los elementos almacenados en él introduciendo en la función el path absoluto del directorio a abrir.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Introducir un directorio que existe en la Base de Datos permitirá obtener una estructura con todos los datos que contiene. - Introducir un directorio que existe pero el usuario no tiene permisos de lectura en él dará lugar a error. - Introducir un directorio que no existe en la Base de Datos dará lugar a error.

TABLA 3.14. REQUISITO FUNCIONAL RF-012

Identificador	RF-012
Título	Función c_readdir
Prioridad	Media
Descripción	Desarrollar la función c_readdir que permitirá obtener el siguiente elemento que esté almacenado en el directorio que se ha pasado en la función c_opendir dada la estructura que recibe la función.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Insertar una estructura vacía dará lugar a error. - Insertar una estructura que no esté vacía permitirá devolver al usuario el siguiente elemento del directorio.

TABLA 3.15. REQUISITO FUNCIONAL RF-013

Identificador	RF-013
Título	Función c_closedir
Prioridad	Media
Descripción	Función c_closedir que cerrará el directorio que hace uso el usuario para moverse por los elementos que estén almacenados en él.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Insertar en la función una estructura vacía dará lugar a error. - Insertar en la función una estructura que se ha obtenido, en un principio, de la función c_opendir permitirá borrar la estructura dando por cerrado el directorio que el usuario ha abierto.

TABLA 3.16. REQUISITO FUNCIONAL RF-014

Identificador	RF-014
Título	Función c_mkdir
Prioridad	Alta
Descripción	Desarrollar la función c_mkdir que permite crear directorios en un directorio padre y al que se le añaden permisos para saber qué usuarios pueden manipularlo tanto para ejecutar, como para escribir o leer en él.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Introducir un path absoluto incorrecto del directorio que se quiere crear dará un error. - Introducir un path absoluto del directorio que se quiere crear dentro de un directorio padre que no existe dará lugar a un error. - Introducir un path absoluto que incluya un fichero regular dará lugar a un error. - Crear un directorio dentro de un directorio padre en el que no se tienen permisos de escritura dará lugar a un error. - Crear un directorio que ya existe en el mismo directorio padre dará lugar a un error. - Crear el directorio raíz dará lugar a error. - Crear un directorio que no existiese antes en un directorio padre que existe en la Base de Datos dará lugar a la creación correcta del directorio.

TABLA 3.17. REQUISITO FUNCIONAL RF-015

Identificador	RF-015
Título	Función c_rmdir
Prioridad	Alta
Descripción	Desarrollar la función c_rmdir que permitirá eliminar un directorio del Sistema de Ficheros creado en la Base de Datos.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Insertar el path absoluto de un directorio que no existe dará lugar a error. - Introducir el path absoluto de un directorio cuyo directorio padre no tenga permisos de escritura para el usuario que ejecuta la función dará lugar a error. - Introducir un directorio que no esté vacío dará lugar a error. - Introducir un path que sea un fichero regular dará lugar a error. - Introducir un directorio existente del que se tenga permisos de escritura en el directorio padre y esté vacío permitirá borrar correctamente el directorio deseado. - Introducir el path del directorio raíz dará lugar a error.

TABLA 3.18. REQUISITO FUNCIONAL RF-016

Identificador	RF-016
Título	Función c_unlink
Prioridad	Alta
Descripción	Desarrollar la función c_unlink que permitirá borrar de la Base de Datos el fichero regular que desee el usuario.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Insertar el path de un fichero regular que no existe dará lugar a error. - Borrar un fichero regular del que no se tengan permisos de escritura en su directorio contenedor dará lugar a error. - Borrar un fichero regular del que se tenga permisos de escritura pero que esté abierto por algún otro usuario dará lugar a error. - Borrar un fichero que tenga abierto el usuario sin previo cierre dará lugar a error. - Borrar un fichero que no tenga ningún proceso abierto y del que se tenga permisos en el directorio contenedor permitirá la ejecución correcta de la función.

TABLA 3.19. REQUISITO FUNCIONAL RF-017

Identificador	RF-017
Título	Integración parcial con MPI
Prioridad	Media
Descripción	Integrar la interfaz desarrollada para acceder a la Base de Datos en MPI utilizando las funciones designadas por el tutor.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	- Abrir un fichero. - Cerrar un fichero. - Escribir en un fichero. - Leer de un fichero.

Requisitos No Funcionales

TABLA 3.20. REQUISITO NO FUNCIONAL RNF-001

Identificador	RNF-001
Título	Manejo de errores
Prioridad	Alta
Descripción	El sistema tratará los errores de la misma forma que POSIX.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	Probar los casos de error en cada función descritos en las pruebas de verificación y compararlo con la documentación proporcionada por POSIX.

TABLA 3.21. REQUISITO NO FUNCIONAL RNF-002

Identificador	RNF-002
Título	Lenguaje de programación de la Interfaz
Prioridad	Alta
Descripción	La interfaz de programación se realizará con el lenguaje de programación C.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	Ninguna.

TABLA 3.22. REQUISITO NO FUNCIONAL RNF-003

Identificador	RNF-003
Título	Base de Datos a utilizar
Prioridad	Alta
Descripción	La Base de Datos que se debe usar para la realización del proyecto debe ser Apache Cassandra.
Fuente	Tutor
Estabilidad	Alta
Pruebas de Verificación	Ninguna.

3.4 Marco regulatorio

En esta última sección del capítulo de Análisis se procederá a explicar las licencias bajo las que se distribuyen cada una de las tecnologías utilizadas para el desarrollo de este proyecto.

En primer lugar, la Base de Datos Apache Cassandra se distribuye bajo una licencia de Software libre, más concretamente bajo la licencia de Apache[43], creado principalmente por la Apache Software Foundation[10]. Esta licencia permite la redistribución y modificación del software siempre que se indique, donde proceda, que el código que ha sido utilizado tenía licencia Apache.

Además, la licencia bajo la que se distribuye el driver de acceso a la Base de Datos, la

empresa Datastax tiene una licencia [44] sobre la que también se permite la modificación y la redistribución del código que un usuario se descargue, al igual que ocurre con la licencia de Apache.

Por último, cabe decir que no existe ninguna otra normativa que sea aplicable al sistema que se quiere diseñar y desarrollar que se está explicando en esta memoria.

Capítulo 4

Diseño

En este capítulo se procederá a describir y explicar las ideas y soluciones que se han decidido realizar en el proyecto. Este capítulo estará formado por una primera parte que constará de la estructura inicial que tendrá la Base de Datos (ver sección 4.1), seguido de los componentes que forman la interfaz basada en POSIX de C que permitirá al usuario acceder a la Base de Datos (ver sección 4.2). Por último, se explicará la interfaz desarrollada que se realizará de MPI sobre Apache Cassandra (ver sección 4.3).

4.1 Estructura de Apache Cassandra para el Sistema de Ficheros

En este apartado se va a explicar cuál ha sido la estructura que se seguirá en la Base de Datos para que se puedan realizar correctamente las operaciones y ejecuciones que se siguen en POSIX para la creación, manipulación y borrado de ficheros y directorios.

Antes de empezar a explicar la estructura utilizada cabe destacar una restricción que la propia Base de Datos impone a la hora de crear tablas y los contenedores o *keyspaces* donde irán estas tablas. Este requisito indica que, para trabajar con las tablas y los *keyspaces* solamente podrán nombrarse con caracteres alfanuméricos y el símbolo "_", de tal forma que, a la hora de implementar la interfaz de POSIX habrá que hacer una serie de consideraciones con respecto al tratamiento de los datos que posteriormente se explicarán.

En primer lugar, se ha decidido que la Base de Datos conste, inicialmente de dos *keyspaces*. Una será *dir* y otra será "_", de tal forma que en el *keyspace* de *dir* se almacenarán todos los directorios que se vayan creando durante la ejecución. Estos directorios serán tablas que seguirán el nombrado "dir.path_absoluto_directorio", donde la segunda

parte del nombre indica el path entero del directorio a manejar. Asímismo, cada tabla de directorio contendrá una serie de campos o columnas básicas que todo directorio tendrá tras ser creado. En este caso, estas tablas contendrán principalmente los campos:

- dirname: Campo que almacena el path absoluto del directorio sin ser modificado, conteniendo los caracteres especiales que no se pueden escribir en el nombre de la tabla asociada.
- "..": Campo que almacena el path absoluto del directorio padre que contiene el presente directorio.
- stat: Campo que almacena todos los metadatos relevantes del directorio al que se refiere y que será utilizado para la función stat que se explicará posteriormente en esta memoria.

Además de estos campos, a medida que el usuario necesite crear otros directorios y/o ficheros regulares, será necesario ampliar el tamaño de esta tabla, por lo tanto, en tiempo de ejecución, estas tablas tenderán a variar su cantidad de columnas, siendo posible añadir nombres de directorio y nombres de fichero que estarán almacenados en ese directorio. En la siguiente ilustración se muestra un ejemplo de esta *keyspace* y lo que puede contener, seguido de un árbol que detalla cómo estarían distribuidos los directorios y los ficheros.

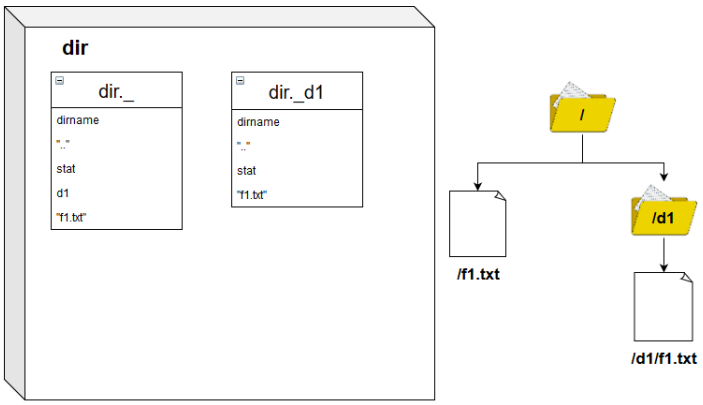


Fig. 4.1. Ejemplo keyspace dir y árbol

Como se puede ver en la Figura 4.1, de forma muy simple, el directorio /d1, que se encuentra dentro del directorio raíz, tiene un fichero con el mismo nombre que el que

tiene almacenado su directorio padre. No obstante, esto no es un problema puesto que, al igual que ocurre en POSIX, en directorios diferentes pueden estar almacenados ficheros con el mismo nombre, gracias a que tienen path absolutos diferentes. Para concluir esta primera *keyspaces* cabe destacar que, inicialmente, el único directorio que estará creado en la Base de Datos será el directorio raíz, directorio que no podrá ser eliminado desde la interfaz de POSIX por seguridad.

La segunda *keyspace* que formará parte de la Base de Datos en su estado inicial será la raíz, donde se almacenarán todos los ficheros regulares que se creen en este directorio. De forma análoga, si en tiempo de ejecución se crean más directorios, se irán creando *keyspaces* en la Base de Datos donde se irán guardando los ficheros regulares asociados a cada uno de ellos y, además, estos directorios crearán sus tablas correspondientes en el *keyspace* dir.

Cuando un directorio se crea se permite la posibilidad de crear ficheros dentro de él si tiene permisos de escritura. Una vez que se crea un fichero, se creará una tabla con los campos:

- Block: Campo que indicará el número de bloque de datos. Tendrá un número extra que será el -1, fila en la que se almacenarán los metadatos del fichero.
- Opened: Campo que indicará la cantidad de usuarios que han abierto el fichero.
- Stat: Campo que almacena todos los metadatos relevantes del fichero al que se refiere y que será utilizado para la función stat que se explicará posteriormente en esta memoria.
- N_bytes: Cantidad de bytes almacenados en el bloque de datos.
- Text: Contenido del bloque de datos.
- N_blocks: Número de bloques totales que tiene el fichero.

Cabe destacar que los campos de opened, stat y n_blocks solamente serán usados para obtener información sobre el fichero en cuestión, es por eso que solamente se escribirán estos datos en una fila de la tabla asociada, mientras que los bloques de datos y los bytes asociados a la cantidad de bytes escritos en ese fichero se irán almacenando en las

diferentes filas o bloques, asignando sus respectivos identificadores de bloque a cada uno. Además, el campo n_bytes de cada fila de la tabla tendrá como máximo valor el tamaño de bloque que se asigne al Sistema de Ficheros. En la siguiente figura se muestra un ejemplo de la estructura que seguirá el Sistema de Ficheros en lo que respecta a los ficheros almacenados en un directorio:

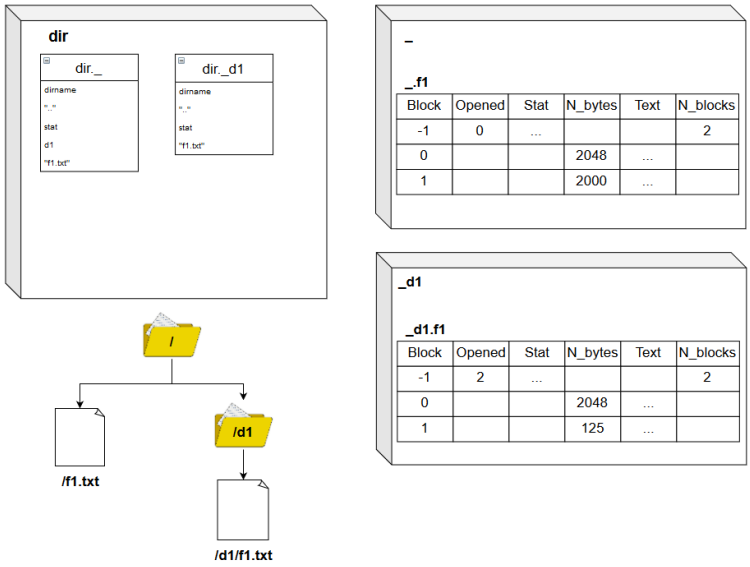


Fig. 4.2. Ejemplo del keyspace y estructura de los ficheros

4.2 Interfaz de acceso basada en POSIX a la Base de Datos

Tras haber explicado el diseño escogido para la Base de Datos en la sección 4.1 se procede a describir el proceso que se ha seguido para diseñar la interfaz que el usuario podrá utilizar para manejar ficheros y directorios.

La implementación de la interfaz que se va a realizar para la Base de Datos Apache Cassandra va a seguir el estándar POSIX, por lo tanto, todas las funciones y métodos que se incluyan en este proyecto seguirán el funcionamiento establecido por este estándar.

Esta biblioteca que se va a diseñar e implementar va a ofrecer funcionalidades para el acceso a ficheros y directorios similar a las del estándar POSIX. Al tratarse de una biblioteca de funciones a nivel de usuario, los nombres de las funciones a desarrollar tendrán el prefijo c_, de tal forma que se pueda diferenciar de las llamadas normales a POSIX.

A continuación, para la descripción de las funciones basadas en POSIX a implementar, para facilitar su entendimiento, se van a clasificar en tres tipos de conjuntos: Primero, aquellas funciones que permiten la manipulación de los ficheros (sección 4.2.1); segundo, las funciones que permiten la manipulación de los directorios (sección 4.2.2); y tercero, las funciones que permiten obtener metadatos de los ficheros y de los directorios existentes (sección 4.2.3).

4.2.1. Manipulación de ficheros regulares

A continuación, se van a explicar las funciones que va a ofrecer la interfaz a desarrollar en lo referente a la manipulación de ficheros regulares.

C_creat

Esta función permite la creación de un fichero regular. En esta función se deberán introducir los diferentes permisos de escritura, lectura y ejecución que tendrá el fichero para los tres tipos de usuarios que pudiesen acceder al fichero. Estos tres grupos son: el usuario, el grupo y otros. Estos permisos se introducirán en formato octal, de igual forma que ocurre en POSIX.

C_open

Esta función permite la apertura de un fichero de la Base de Datos para su posterior manipulación ya sea para leer o escribir en él. El método recibe, al igual que en POSIX, varios argumentos, que son el path absoluto del fichero, y los flags o modos de apertura del fichero. Con respecto a este último campo, se ha dado soporte a los siguientes modos:

- O_CREAT: permite la creación de un fichero si éste no existiese en la Base de Datos. Este fichero se crearía con permisos de lectura, escritura y ejecución para todos los tipos de usuarios por defecto. Posteriormente, se abriría el fichero para el usuario que lo creó.
- O_TRUNC: permite el truncamiento del fichero si, además, el usuario tiene permisos de escritura sobre él. En caso de tener permisos, se borrará toda la información

guardada en el fichero.

- O_APPEND: modo que permite mover, de forma local, el puntero de posición del fichero al final del mismo.
- O_RDWR: modo que indica que el fichero asociado se va a leer o escribir mientras que esté abierto.

Todos estos modos anteriormente descritos funcionan de manera local, es decir, no tiene ningún efecto para los demás usuarios que estén accediendo al fichero de manera simultánea, por lo que el proceso que esté accediendo al fichero deseado será el que se vea afectado por estos modos soportados.

C_read

Función que leerá el fichero abierto anteriormente. Para poder leerlo, el método recibe varios parámetros que son el descriptor de fichero asociado, el buffer donde se guardarán los datos y la cantidad de bytes que se quieren leer del fichero.

C_write

Función que escribirá el fichero abierto anteriormente. Para poder escribir en él, el método recibe varios parámetros que son el descriptor de fichero asociado, el buffer donde están los datos que se quieren almacenar en el fichero y la cantidad de bytes que se quieren escribir en él.

C_close

Función que permite el cierre del fichero si ha sido abierto anteriormente por el proceso.

C_unlink

Función que permite borrar el fichero de la Base de Datos si se tienen permisos de escritura sobre el fichero asociado y, además, no hay ningún otro usuario que esté acce-

diendo al mismo fichero que se quiere eliminar.

C_lseek

Función que permite sobre un fichero, de manera local, modificar el puntero de posición para su posterior escritura o lectura de la información almacenada en él. Para poder mover el puntero, el método recibe diferentes argumentos, que son el descriptor de fichero obtenido tras su apertura, el número de bytes que se quiere mover (offset) y, por último, desde dónde se quiere empezar a mover el puntero (whence). Este último parámetro tiene tres diferentes modos o tipos:

- SEEK_SET: Modo que cambia el puntero al principio del fichero.
- SEEK_CUR: Modo que deja el puntero en la posición actual que tuviera.
- SEEK_END: Modo que cambia el puntero al final del fichero.

De igual forma que en la función C_open, estas operaciones de modificación del puntero se realizan de manera local afectando solamente al proceso que maneja el fichero.

4.2.2. Manipulación de directorios

A continuación, se van a explicar las funciones que va a ofrecer la interfaz a desarrollar en lo referente a la manipulación de directorios.

C_mkdir

Esta función permite la creación de un directorio. En esta función se deberán introducir los diferentes permisos de escritura, lectura y ejecución que tendrá el directorio para los tres tipos de usuarios que pudiesen acceder a él. Estos tres grupos son: el usuario, el grupo y otros. Estos permisos se introducirán en formato octal, de igual forma que ocurre en POSIX.

C_rmdir

Función que eliminará de la Base de Datos el directorio que indique el usuario introduciendo el path absoluto del mismo. Este directorio se eliminará si no existe ningún elemento dentro de él y, además, se tienen permisos en el directorio para ser modificado.

C_opendir

Función que permitirá abrir el directorio que el usuario indique en el argumento de la función para su posterior lectura de elementos. Se podrá leer el directorio deseado si se tienen permisos de lectura.

C_readdir

Función que permitirá leer el directorio que haya abierto anteriormente el usuario para poder obtener los elementos almacenados en él.

C_closedir

Función que cerrará el directorio que el usuario haya abierto anteriormente.

4.2.3. Obtención de metadatos

Por último, se van a explicar los métodos que permitirán obtener la información o datos de los ficheros regulares y directorios que estén almacenados en la Base de Datos.

C_stat

Función que permite obtener los metadatos de los ficheros y de los directorios que se encuentren almacenados en la Base de Datos. Para saber esta información el usuario tendrá que introducir el path absoluto del fichero o del directorio y una estructura donde se guardarán los datos asociados a estos elementos.

C_fstat

Función que permite obtener los metadatos de los ficheros. Para que reciba esta información, el usuario deberá tener el fichero abierto para poder usar esta función pues tiene como argumento un descriptor de fichero abierto junto con una estructura en la que se debe almacenar la información asociada.

4.3 Interfaz MPI-IO sobre Apache Cassandra

En esta última sección se va a describir la construcción que se ha seguido para añadir la Base de Datos Apache Cassandra sobre la biblioteca que se desarrolle en MPI-IO para poder realizar las diferentes operaciones de manejo de ficheros sobre Cassandra.

Esta interfaz a desarrollar se incluirá en Romio, dentro de ADIO, que es una interfaz abstracta. En la siguiente figura se muestra la integración de la Base de Datos dentro de MPI-IO:

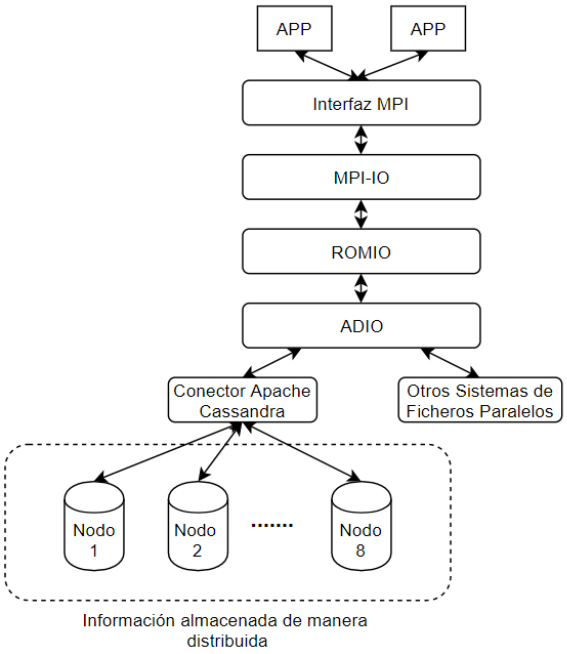


Fig. 4.3. Integración de Apache Cassandra en Romio

Para la implementación que se debe realizar de ADIO, para la Base de Datos Apache Cassandra, consiste en seguir el diseño que se ha realizado para algunas de las funciones que se han utilizado para el diseño de la interfaz basada en POSIX descritas en la sección

4.2. En las siguientes líneas se expondrán las funciones que se tendrán que implementar para poder manejar ficheros (sección 4.3.1)

4.3.1. Funciones de manipulación de ficheros

A continuación, se describirán brevemente las funciones que ofrece MPI-IO y que van a ser implementadas para este proyecto:

- MPI_File_Open: Función encargada de abrir un fichero. Todas las llamadas que se realicen a esta función son colectivas, es decir, que cada uno de los procesos que estén en el comunicador necesitan hacer esta llamada para poder abrir el fichero. Los modos de apertura que aceptará esta función serán:
  - MPI_MODE_CREAT: Permite la creación de un fichero si no ha sido creado anteriormente.
  - MPI_MODE_APPEND: Modo que permite la apertura de un fichero de tal forma que se empiece a leer o escribir desde el final del mismo. Este modo solo se puede ejecutar en un proceso.
  - MPI_MODE_RDWR: Modo que permite que el proceso que abra el fichero pueda tanto leer como escribir.
  - MPI_MODE_TRUNC: Modo que permite el truncamiento del fichero vaciando los datos que tenga almacenados.
- MPI_File_Close: Función colectiva, al igual que la anterior, que permite cerrar un fichero. De esta forma, todos los procesos esperan a que aquellos procesos que abrieron el fichero lo cierren.
- MPI_File_Read: Función de lectura que permite a los procesos obtener la información almacenada en los ficheros.
- MPI_File_Write: Función de escritura que permite almacenar la información deseada en el fichero.

Las dos últimas funciones pueden ser bloqueantes o no bloqueantes, es decir, que los procesos pueden bloquearse o no hasta que termine de ejecutarse la operación; y también

pueden ser colectivas o no colectivas, es decir, las colectivas indican que los procesos que abran un fichero tienen que realizar una operación de escritura o de lectura, mientras que las no colectivas no tienen por qué ejecutarlas cada uno de los procesos.

Capítulo 5

Implementación y Desarrollo

En este capítulo de la memoria se van a explicar las implementaciones que se han llevado a cabo en el sistema a desarrollar para cumplir con los requisitos estipulados en la sección 3.3 del presente documento. En primer lugar, se va a dar una explicación concisa de la información que se tiene que crear en la Base de Datos inicialmente (sección 5.1), seguido de una descripción detallada de la implementación realizada para la interfaz que dará soporte a la Base de Datos que cumplirá la función de Sistema de Ficheros (sección 5.2). Y, por último, se dará una explicación de la interfaz de MPI-IO para que pueda soportar la Base de Datos Apache Cassandra (sección 5.3).

5.1 Construcción inicial de Apache Cassandra

Esta sección tratará de explicar cómo se ha procedido a manejar la Base de Datos para poder implementar después la interfaz de las funciones. En este caso, el funcionamiento es muy simple. Tal y como se ha comentado anteriormente, solamente se necesitan dos "keyspaces", una para almacenar todos los directorios que se creen con sus respectivos elementos que se quieran almacenar y los metadatos de los directorios creados, junto con otra "keyspace" que será el directorio raíz ("_").

En un principio, se utilizaron estos espacios de trabajo de tal forma que su factor de replicación fuese 1, esto es, manteniendo una sola copia en el cluster, pero para la realización de las pertinentes pruebas que se definirán más adelante, se utilizará tanto factor de replicación 1 como factor de replicación 3 para comprobar tasas de trasferencia. Por lo tanto, las sentencias utilizadas para la construcción inicial de la Base de Datos son:

1. Creación “keyspace” dir: CREATE KEYSPACE IF NOT EXISTS dir WITH replication = ’class’: ’SimpleStrategy’, ’replication_factor’: ’1’ ;

2. Creación “keyspace” del directorio raíz: CREATE KEYSPACE IF NOT EXISTS “_” WITH replication = ’class’: ’SimpleStrategy’, ’replication_factor’: ’1’ ;

3. Creación de la tabla del directorio raíz: CREATE TABLE IF NOT EXISTS dir.“_” (dirname text,".."text, uid text, mode text, stat text, PRIMARY KEY(dirname));

4. Inserción de los metadatos iniciales del directorio raíz: INSERT INTO dir.“_” (dirname,“..”,uid,mode,stat) VALUES (’/’,’/’,’0’,’0755’,’stat’);

5.2 Implementación de la Interfaz POSIX

La implementación de la interfaz se ha realizado en el lenguaje de programación C, mientras que Apache Cassandra se encuentra escrito en Java, por lo que ha sido necesario hacer uso de un conector o driver auxiliar que contiene una interfaz de llamadas específica para poder interactuar con la Base de Datos.

Este driver [40], como se ha explicado anteriormente, pertenece a la empresa Datastax, y contiene una amplia variedad de métodos y funciones a tener en cuenta para la correcta conexión del usuario con Cassandra y poder usarlo correctamente. No obstante, cabe destacar que este driver hace uso exclusivamente del lenguaje que provee la Base de Datos (CQL3) y siguiendo el protocolo nativo que ofrece Cassandra para poder enviar y recibir los datos acorde a lo que se especifique en ese protocolo [45].

La siguiente figura muestra la interacción entre los componentes de la parte usuario con Cassandra, haciendo uso del driver necesario para la conexión:

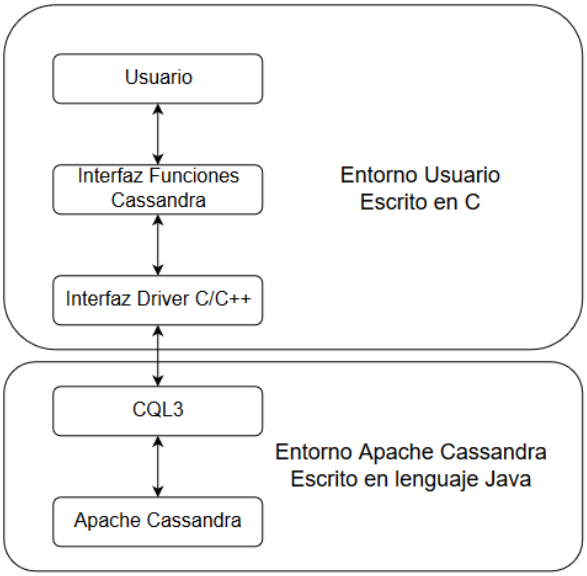


Fig. 5.1. Interacción entre los componentes

5.2.1. Estructura de la aplicación

A continuación, se va a explicar cómo se ha estructurado la implementación de la interfaz con el objetivo de tener un código estructurado. En la siguiente figura se muestra la estructura de los directorios que se ha seguido:

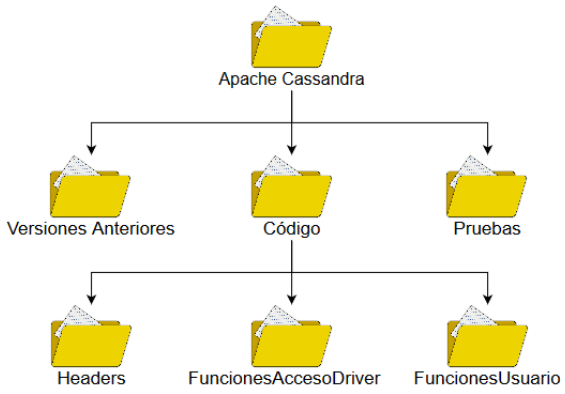


Fig. 5.2. Estructura de los directorios de la aplicación

Con respecto a la figura anterior, es importante explicar los directorios que se encuentran dentro de la carpeta “Código”, que es aquella en la que se encuentra el grueso de la aplicación a desarrollar:

- Headers: En este directorio se encuentran las cabeceras de las funciones que formarán parte de la interfaz de acceso a la Base de Datos que podrá utilizar el usuario. Además, también se encuentra la definición de las funciones auxiliares que se utilizarán para conectarse y hacer las peticiones pertinentes a la Base de Datos.

- FuncionesAccesoDriver: En este directorio se encuentran los códigos de aquellas funciones que han sido necesarias para establecer la conexión, el envío y la recepción de datos con la Base de Datos. Estos métodos nos son usados por el usuario sino que son métodos auxiliares a las funciones principales del directorio FuncionesUsuario.

- FuncionesUsuario: En este directorio se encuentran las implementaciones de las funciones que utilizarán los usuarios para hacer uso de Cassandra.

A parte de estos directorios anteriormente mencionados, también cabe destacar el directorio de Pruebas, directorio en el que se guarda tanto el main, que será el fichero donde se realizarán las llamadas de la interfaz implementada, junto con otros ficheros que servirán de apoyo para las posteriores pruebas de rendimiento del sistema. Por otro lado, cuando se producen cambios importantes en el código, todos los directorios anteriormente mencionados, se guardarán en el directorio de "Versiones Anteriores" indicando la última fecha de modificación, de tal forma que se tenga un código organizado.

5.2.2. Pseudocódigo de las funciones de manejo de ficheros

Tras explicar la estructura que se ha seguido, se van a mostrar los pseudocódigos de las implementaciones utilizadas para el desarrollo de este sistema. En esta parte, más concretamente, se mostrarán los pseudocódigos de las funciones de tratamiento de ficheros.

En primer lugar se muestra la función de creación de ficheros, c_creat:

Algorithm 1: Función c_creat Parte 1

Input: const char *pathname, mode_t mode

Output: int cod_error

```
1  if pathname no es absoluto then
2      return error
   /* iniciado == 0                                                    */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  Separar nombre del directorio del nombre del fichero.
7  Cambiar los caracteres especiales por _
8  if No existe directorio en Cassandra then
9      return error
10 if No se tienen permisos de escritura en el directorio then
11     return error
12 if El fichero ya existe then
13     return error
14 Obtención uid, gid, hora de creación, hora de modificación e inserción en
     metadatos
15 Inserción en metadatos del tipo de objeto, el tamaño total y los permisos
   /* Crear columna con nombre de fichero en el directorio contenedor  */
16 ALTER TABLE dir.path_directorio ADD nombre_fichero int;
   /* Creación de la tabla del fichero                                 */
17 CREATE TABLE path_directorio.nombre_fichero
18 (block int, opened int, uid text, mode text, stat text,
19 n_blocks int, n_bytes int,
20 data text, PRIMARY KEY(block));
```

Algorithm 2: Función c_creat Parte 2

```
/* Inserción metadatos en el fichero                                    */
21 INSERT INTO path_directorio.nombre_fichero
    (block,opened,stat,n_blocks,n_bytes,uid,mode)
22 VALUES (-1,0,stat,1,0,uid,mode);
23 return OK
```

La siguiente función corresponde a la apertura de un fichero, c_open:

Algorithm 3: Función c_open Parte 1

Input: const char *pathname, int flags

Output: int fd/cod_error

```
1  if pathname no es absoluto then
2      return error
   /* iniciado == 0                                                     */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  Separar nombre del directorio del nombre del fichero.
7  Cambiar los caracteres especiales por _
8  if No existe directorio en Cassandra then
9      return error
10 if No se tienen permisos de escritura en el directorio then
11     return error
12 if fichero ya abierto then
13     return error
14 if El fichero no existe then
15     if flags no contiene O_CREAT then
16         return error
17     else
18         no_existe = 1
```

Algorithm 4: Función c_open Parte 2

```
19 if flags contiene O_CREAT then
20     if flags contiene O_TRUNC then
21         if no_existe == 1 then
22             Se crea el fichero
23         if Se tienen permisos de escritura then
               /* Se trunca el fichero                                  */
24             Obtención de metadatos del fichero
25             TRUNCATE TABLE path_directorio.nombre_fichero
26             Inserción metadatos en el fichero truncado
27     else
28         if no_existe == 1 then
29             Se crea el fichero
30 else
31     if Se tienen permisos de escritura then
           /* Se trunca el fichero                                      */
32         Obtención de metadatos del fichero
33         TRUNCATE TABLE path_directorio.nombre_fichero
34         Inserción metadatos en el fichero truncado
35 Se busca el primer hueco libre de descriptores de ficheros abiertos del proceso.
   /* Se aumenta en 1 el número de procesos que tienen abiertos el fichero en
      la Base de Datos                                                  */
36 SELECT opened, n_blocks FROM path_directorio.nombre_fichero WHERE
     block = -1
37 UPDATE path_directorio.nombre_fichero SET opened = opened+1 WHERE
     block = -1
38 if flags contiene O_APPEND then
39     punteroProceso=FinalFichero
40 else
41     punteroProceso=0
42 return fd
```

Tras la apertura del fichero, se muestra la forma de cerrarlo, utilizando la función c_close:

Algorithm 5: Función c_close

Input: int fd

Output: int cod_error

```
1  if fd <0 then
2      return error
   /* iniciado == 0                                                    */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  if El fd tiene un fichero asociado abierto then
       /* Se disminuye en 1 el número de procesos que tienen abiertos el
          fichero en la Base de Datos                                   */
7      SELECT opened, n_blocks FROM path_directorio.nombre_fichero WHERE
        block = -1
8      UPDATE path_directorio.nombre_fichero SET opened = opened-1 WHERE
        block = -1
9      Se elimina la información del fichero del proceso.
10     return OK
11 else
12     return error
```

Si el usuario desea eliminar de la Base de Datos algún fichero, tendrá que hacer uso de la función c_unlink:

Algorithm 6: Función c_unlink

Input: int fd

Output: int cod_error

```
1  if pathname no es absoluto then
2      return error
   /* iniciado == 0                                                    */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  if pathname es un fichero then
7      if No existe directorio en Cassandra then
8          return error
9      if fichero ya abierto then
10         return error
11     if No se tienen permisos de escritura en el directorio then
12         return error
13     if Fichero abierto por el proceso actual then
14         return error
15     if Fichero abierto por otro proceso then
16         return error
       /* Se procede a borrar el fichero de la BBDD                    */
17     DROP TABLE path_directorio.nombre_fichero
18     ALTER TABLE dir.path_directorio DROP nombre_fichero
19     return OK
20 else
21     return error
```

Si, por ejemplo, el usuario tiene algún fichero abierto y desea modificar el puntero asociado a ese fichero, podrá hacer uso de la función c_lseek:

Algorithm 7: Función c_lseek

```
Input: int fd, off_t offset, int whence
Output: off_t size
1  if fd <0 then
2      return error
   /* iniciado == 0                                                    */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  if fd abierto then
7      if whence == SEEK_CUR then
8          PunteroFicheroProceso[fd] = PunteroFicheroProceso[fd]+offset
9      else if whence == SEEK_SET then
10         PunteroFicheroProceso[fd] = offset
11     else
12         PunteroFicheroProceso[fd] =
             (NumeroBloquesFichero[fd]*TamañoBloque)+offset
13     return PunteroFicheroProceso[fd]
14 else
15     return error
```

A continuación, se muestran las dos funciones que permiten la lectura y escritura de un fichero, muy importantes en esta parte del manejo de ficheros, puesto que sin estos métodos no se podría comprobar el funcionamiento de la Base de Datos con respecto a las tasas de transferencia que son los datos que se quieren manejar para determinar su eficacia. Estas funciones son c_read y c_write, las cuales se describirán sus pseudocódigos a continuación:

Algorithm 8: Función c_read

Input: int fd, off_t offset, int whence

Output: ssize_t size

```
1  if fd <0 && count <0 then
2      return error
   /* iniciado == 0                                                    */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  if fd cerrado then
7      return error
8  if No se tienen permisos de lectura en el fichero then
9      return error
10 bytesPorLeer = count
11 while bytesPorLeer != 0 do
12     BloquePuntero = PunteroFicheroProceso[fd]/TamañoBloque
13     PosBloque = PunteroFicheroProceso[fd] %TamañoBloque
       /* Obtención del bloque de datos a leer de la Base de Datos      */
14     SELECT n_bytes, data FROM path_directorio.nombre_fichero where block =
        BloquePuntero; if bytesPorLeer >TamañoBloque then
15         Copiar bytes al buffer hasta tamaño de bloque
16         BytesPorLeer -= BytesLeidos
17     else
18         if BytesPorLeer + PosBloque >= TamañoBloque then
19             Copiar bytes hasta tamaño del bloque al buffer
20             BytesPorLeer -= BytesLeidos
21         else
22             Copiar bytes restantes al buffer
23             BytesPorLeer -= BytesLeidos
24 return count - bytesPorLeer
```

Después de explicar la función c_read, se muestra el pseudocódigo de la función c_write:

Algorithm 9: Función c_write Parte 1

Input: int fd, off_t offset, int whence

Output: ssize_t size

```
1  if fd <0 && count <0 then
2      return error
   /* iniciado == 0                                                    */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  if fd cerrado then
7      return error
8  if No se tienen permisos de escritura en el fichero then
9      return error
10 bytesPorEscribir = count
```

Algorithm 10: Función c_write Parte 2

```
11 while bytesPorLeer != 0 do
12     BloquePuntero = PunteroFicheroProceso[fd]/TamañoBloque
13     PosBloque = PunteroFicheroProceso[fd] %TamañoBloque
14     if posBloque != 0 && bytesPorEscribir <TamañoBloque then
           /* Obtención del bloque de datos a escribir de la Base de Datos si
              es menor del tamaño de bloque. En caso contrario se
              sobreescribirá el bloque entero                                  */
15         SELECT n_bytes, data FROM path_directorio.nombre_fichero where
            block = BloquePuntero
16     if bytesPorEscribir >TamañoBloque then
           /* Escribir bytes a Base de Datos hasta tamaño de bloque            */
17         ; INSERT INTO path_directorio.nombre_fichero (n_bytes,data,block)
            VALUES (Bytes,buffer,NBloque)
18         BytesPorEscribir -= BytesEscritos
19     else
20         if BytesPorLeer + PosBloque >= TamañoBloque then
               /* Escribir bytes hasta tamaño del bloque al buffer             */
21             ; INSERT INTO path_directorio.nombre_fichero (n_bytes,data,block)
                VALUES (Bytes,buffer,NBloque)
22             BytesPorEscribir -= BytesEscritos
23         else
               /* Copiar bytes restantes al buffer                             */
24             ; INSERT INTO path_directorio.nombre_fichero (n_bytes,data,block)
                VALUES (Bytes,buffer,NBloque)
25             BytesPorEscribir -= BytesEscritos
26     if PunteroFicheroProceso[fd] + BytesEscritos ==
        BloquesFichero[fd]*TamañoBloque then
27         Crear bloque nuevo en Base de Datos.
28 return count - bytesPorEscribir
```

5.2.3. Pseudocódigo de las funciones de manejo de directorios

A continuación, se mostrarán los métodos que permiten el manejo de directorios, ya sea creación, borrado o lectura de los elementos que estén en los directorios. La primera que se va a mostrar es la de creación de directorios, c_mkdir:

Algorithm 11: Función c_mkdir Parte 1

Input: const char *pathname, mode_mode

Output: int cod_error

```
1  if pathname no es absoluto then
2      return error
   /* iniciado == 0                                                    */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  Cambiar los caracteres especiales por _
7  if Pathname es un fichero then
8      return error
9  if No existe el directorio padre en Cassandra then
10     return error
11 if Ya existe el directorio en Cassandra then
12     return error
13 if No se tienen permisos de escritura en el directorio padre then
14     return error
15 Obtención uid, gid, hora de creación, hora de modificación e inserción en
     metadatos
16 Inserción en metadatos del tipo de objeto, el tamaño total y los permisos
   /* Crear columna con nombre de directorio en el directorio contenedor */
17 ALTER TABLE dir.path_directorio_padre ADD nombre_directorio int
   /* Creación de la "keyspace" del directorio                         */
18 CREATE KEYSPACE path_directorio WITH replication =
     {'class':'SimpleStrategy','replication_factor':'3'}
```

Algorithm 12: Función c_mkdir Parte 2

```
19 CREATE TABLE dir.path_directorio (dirname text, ".." text, uid text, mode text,
     stat text, PRIMARY KEY (dirname))
   /* Inserción metadatos del directorio                                    */
20 INSERT INTO dir.path_directorio VALUES path, parent_dir, uid, mode, stat
21 return OK
```

Después de mostrar la creación de un directorio, se procede a mostrar la eliminación de un directorio con la función c_rmdir:

Algorithm 13: Función c_rmdir Parte 1

```
   Input: const char *pathname
   Output: int cod_error
1  if pathname no es absoluto then
2      return error
   /* iniciado == 0                                                        */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  Cambiar los caracteres especiales por _
7  if Pathname es un fichero then
8      return error
9  if No existe el directorio padre en Cassandra then
10     return error
11 if No existe el directorio en Cassandra then
12     return error
13 if No está vacío el directorio en Cassandra then
14     return error
15 if No se tienen permisos de escritura en el directorio then
16     return error
```

Algorithm 14: Función c_rmdir Parte 2

```
/* Borrado del directorio de la Base de Datos                    */
17 DROP TABLE dir.path_directorio
18 ALTER TABLE dir.path_directorio_padre DROP directorio
19 DROP KEYSPACE path_directorio
20 return OK
```

A continuación, se mostrarán las funciones de c_opendir, c_readdir y c_closedir que permiten obtener información de los elementos almacenados en el directorio que decida abrir el usuario:

Algorithm 15: Función c_opendir Parte 1

Input: const char *pathname

Output: DIR directorio

```
1  if pathname no es absoluto then
2      return error
   /* iniciado == 0                                                */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  Cambiar los caracteres especiales por _
7  if Pathname es un fichero then
8      return error
9  if No existe el directorio en Cassandra then
10     return error
11 if No se tienen permisos de lectura en el directorio then
12     return error
13 if El directorio ya está abierto por el proceso then
14     return error
```

Algorithm 16: Función c_opendir Parte 2

15 Se almacena el directorio en el registro de directorios abiertos por el proceso

`/* Se obtienen los elementos del directorio */`

16 SELECT * FROM system_schema.columns WHERE keyspace_name = dir and table_name = path_directorio and type = int

17 **while** *No haya más elementos en el directorio* **do**

18 Guardar en lista el nombre del elemento y su tipo

19 return Directorio DIR

Algorithm 17: Función c_readdir

Input: DIR *directorio

Output: struct dirent dir

1 **if** *directorio está vacío* **then**

2 return error

`/* Obtener siguiente elemento del directorio */`

3 dir.d_name = directorio->filepos+1; **if** *dir.d_name es fichero* **then**

4 dir.d_type = DT_REG

5 **else**

6 dir.d_type = DT_DIR

7 return directorio

Algorithm 18: Función c_closedir

Input: DIR *directorio

Output: int cod_error

1 **if** *directorio está vacío* **then**

2 return error

`/* Cerrar el directorio abierto por el proceso */`

3 Eliminación del directorio de la lista de directorios abiertos

4 return OK

5.2.4. Pseudocódigo de las funciones de obtención de metadatos

Por último, se mostrarán las funciones que permitirán a los usuarios obtener la información relativa a los directorios y a los ficheros regulares que desee. Estas funciones son c_stat y c_fstat que se mostrarán a continuación:

Algorithm 19: Función c_stat Parte 1

Input: const char *pathname, struct stat *buf

Output: int cod_error

```
1  if pathname no es absoluto then
2      return error
   /* iniciado == 0                                                    */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  if pathname es un fichero then
7      if el directorio contenedor no existe then
8          return error
9      else
10         if fichero existe then
               /* Obtención metadatos fichero                          */
11             SELECT stat FROM path_directorio.nombre_fichero WHERE block
                = -1
12             Copiar stat a estructura buf
13             return OK
14         else
15             return error
```

Algorithm 20: Función c_stat Parte 2

```
16 else
17     if el directorio no existe then
18         return error
19     else
           /* Obtención metadatos directorio                    */
20         SELECT stat FROM dir.nombre_directorio WHERE dirname =
             nombre_directorio
21         Copiar stat a estructura buf
22         return OK
```

Algorithm 21: Función c_fstat

Input: int fd, struct stat *buf

Output: int cod_error

```
1  if fd <0 then
2      return error
   /* iniciado == 0                                             */
3  if No se ha iniciado antes la sesión con la Base de Datos then
4      Iniciar conexión con la Base de Datos.
5      iniciado = 1
6  if Fichero abierto por el proceso actual then
7      return error
   /* Obtención metadatos fichero                               */
8  SELECT stat FROM path_directorio.nombre_fichero WHERE block = -1
9  Copiar stat a estructura buf
10 return OK
```

5.3 Implementación de la Interfaz de MPI-IO

Para empezar, debido a que Cassandra está escrito en Java, para poder implementar la interfaz en MPI-IO en C, es necesario hacer uso del conector o driver del que se ha comentado anteriormente, en la sección 1.1. En la siguiente figura se muestra las diferentes

interacciones que se realizan entre las diferentes bibliotecas para poder acceder a la Base de Datos

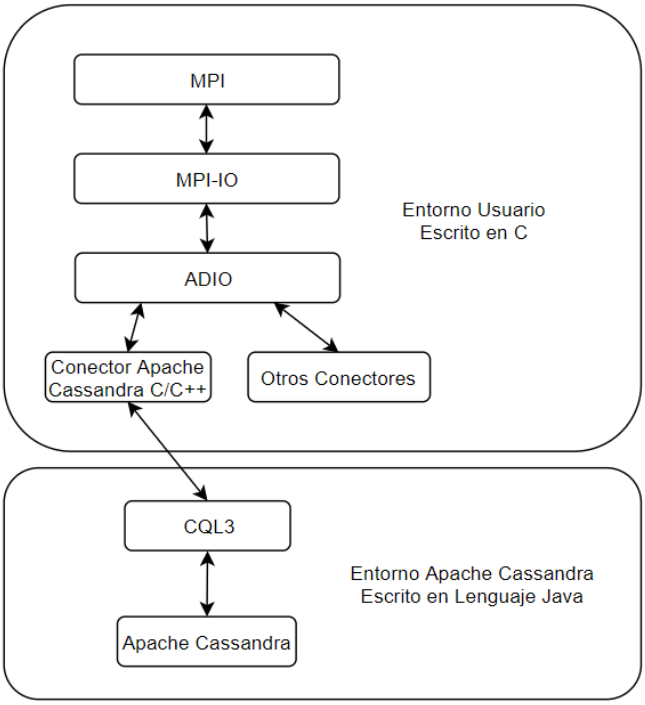


Fig. 5.3. Interacción entre los componentes en MPI

A continuación, se explicarán las implementaciones que se han seguido para las funciones descritas en el Capítulo 4 del Diseño (ver sección 4.3).

Función de apertura de un fichero

Como se ha explicado anteriormente, la función de apertura permitirá únicamente los siguientes modos de apertura:

- MPI_MODE_CREAT: Permite la creación de un fichero.
- MPI_MODE_APPEND: Modo que permite la apertura de un fichero de tal forma que se empiece a leer o escribir desde el final del mismo..
- MPI_MODE_RDWR: Modo que permite que el proceso que abra el fichero pueda tanto leer como escribir.
- MPI_MODE_TRUNC: Modo que permite el truncamiento del fichero vaciando los datos que tenga almacenados.

Al igual que la interfaz desarrollada para POSIX, la semántica en MPI-IO será similar. Si un usuario quiere crear un fichero, necesitará indicar en la función de apertura, que desea crearlo si no existe con el modo apropiado. De igual forma, si quiere leerlo o escribirlo también tendrá que indicarlo con el modo necesario. Si, por otro lado, necesita borrar su contenido, deberá truncarlo o si desea empezar a leer o escribir desde el final del fichero, tendrá que utilizar el modo MPI_MODE_APPEND.

Función de lectura de un fichero

Para la implementación de esta función se ha utilizado un único tipo de puntero de entre todos los que ofrece MPI, que consiste en utilizar un puntero individual por cada proceso que desee acceder a un fichero cualquiera. Para ello se utiliza la función MPI_File_Seek que permite la modificación de este puntero. Si se llegase al final del fichero, la función devolvería un 0 indicando que ha leído hasta el final. Para esta función de lectura, el usuario tendrá que indicar la cantidad de bytes que desea leer y el buffer donde desea obtenerlo y, como resultado, la función terminará de ejecutar cuando haya leído todos los bytes indicados o cuando llegue al final del fichero.

Función de escritura de un fichero

De igual forma que con la interfaz de POSIX, si un usuario desea escribir en un fichero, nada más abrirlo, su puntero se situará al principio, al igual que ocurre si un usuario decide usar la función de lectura. Para esta función de escritura, el usuario tendrá que indicar la cantidad de bytes que desea escribir y el buffer donde están guardados los datos que la Base de Datos debe guardar y, como resultado, la función terminará de ejecutar cuando haya escrito todos los bytes indicados.

Función de cerrado de un fichero

Para implementar esta función lo que se realiza es indicar el descriptor del fichero que el proceso ha abierto anteriormente. No obstante, si no se hubiese abierto antes ningún fichero por el proceso que utiliza esta función, se notificaría con un error.

Capítulo 6

Verificación, validación y evaluación

En este capítulo se procederá a detallar la verificación, validación y evaluación del proyecto que se está explicando en este trabajo. Para ello, primero se realizará la verificación del funcionamiento del sistema (ver sección 6.1), comprobando que se cumplan todos los requisitos establecidos por el usuario en el capítulo de análisis (ver capítulo 3). Por último, se presentará la evaluación del sistema (ver sección 6.2) con una comparativa de los rendimientos de la interfaz que se centrará en:

- Comparativa de tasas de transferencia de Apache Cassandra.
- Comparativa de tasas de transferencia entre Apache Cassandra con HDFS.
- Comparativa de tasas de transferencia entre Apache Cassandra con MPI-IO.

6.1 Verificación y validación

Para determinar si el sistema desarrollado cumple con lo establecido por el cliente (en este caso es el tutor del proyecto), se van a comprobar los requisitos propuestos mediante pruebas de verificación (ver sección 6.1.1). Además, se va a determinar el correcto funcionamiento de la interfaz y de su integración en MPI mediante la realización de las pruebas de validación (ver sección 6.1.2).

6.1.1. Pruebas de verificación

A continuación, se van a comprobar que el sistema funciona correctamente si cumple con los requisitos que se han definido en el capítulo 3, más concretamente la sección 3.3.

Para determinar si se cumplen los requisitos se va a seguir como plantilla la siguiente tabla:

TABLA 6.1. PLANTILLA PARA LAS PRUEBAS DE VERIFICACIÓN

Identificador	Sirve para identificar de forma unívoca la prueba. Seguirán la estructura PVE-XX, siendo PVE el acrónimo de Prueba de Verificación y XX serán dos dígitos.
Requisito relacionado	Identificador del requisito funcional o funcional que se verifica en la prueba que se realice.
Descripción	Explicación de la prueba de verificación que se está realizando.
Precondiciones	Condiciones previas a la realización de la prueba.
Procedimiento	Pasos a realizar para realizar la prueba.
Postcondiciones	Condición en la que se encuentra el sistema después de la prueba.
Éxito	Campo que indica si la prueba ha resultado satisfactoria o no.

Después de haber definido la plantilla a seguir, las pruebas de verificación que se han realizado son las siguientes:

TABLA 6.2. PRUEBA DE VERIFICACIÓN PVE-01

Identificador	PVE-01
Requisito relacionado	RF-001 RF-002 RF-003 RF-004 RF-005 RF-006 RF-007 RF-008 RF-009 RF-010 RF-011 RF-012 RF-013 RF-014 RF-015 RF-016 RNF-001 RNF-002 RNF-003
Descripción	La nueva interfaz se conecta a la Base de Datos.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos.
Procedimiento	1. El usuario realiza una llamada a una de las funciones que dispone la interfaz.
Postcondiciones	1. El proceso abre una comunicación con la Base de Datos si anteriormente no se ha conectado.
Éxito	Sí.

TABLA 6.3. PRUEBA DE VERIFICACIÓN PVE-02

Identificador	PVE-02
Requisito relacionado	RF-001 RF-002 RF-003 RF-004 RF-005 RF-006 RF-007 RF-008 RF-009 RF-010 RF-011 RF-012 RF-013 RF-014 RF-015 RF-016 RNF-001 RNF-002 RNF-003
Descripción	La nueva interfaz se desconecta de la Base de Datos cuando termina su ejecución.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos.
Procedimiento	1. El proceso termina su ejecución.
Postcondiciones	1. El proceso cierra la conexión con la Base de Datos.
Éxito	Sí.

TABLA 6.4. PRUEBA DE VERIFICACIÓN PVE-03

Identificador	PVE-03
Requisito relacionado	RF-001 RF-003 RNF-001 RNF-002 RNF-003
Descripción	Se crea un fichero en la Base de Datos.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El proceso se ha conectado con la Base de Datos. 4. El fichero a introducir no existe.
Procedimiento	1. El usuario llama a la función de creación de fichero. 2. El usuario introduce el path absoluto del fichero junto con el nombre que tendrá en el Sistema. 3. El usuario introduce los permisos del fichero.
Postcondiciones	1. El fichero se crea en la Base de Datos sin errores.
Éxito	Sí.

TABLA 6.5. PRUEBA DE VERIFICACIÓN PVE-04

Identificador	PVE-04
Requisito relacionado	RF-001 RF-003 RF-004 RNF-001 RNF-002 RNF-003
Descripción	Apertura de un fichero.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El proceso se ha conectado con la Base de Datos. 4. El fichero a introducir existe.
Procedimiento	1. El usuario llama a la función de creación de fichero. 2. El usuario introduce el path absoluto del fichero que desea abrir y los flags con los que abrirá el fichero.
Postcondiciones	1. Se aumenta en uno el número de ficheros que tienen abierto el fichero en la Base de Datos. 2. El proceso obtiene un identificador de fichero para poder manejarlo. 3. El proceso almacena los datos del fichero en una posición de la lista de ficheros abiertos.
Éxito	Sí.

TABLA 6.6. PRUEBA DE VERIFICACIÓN PVE-05

Identificador	PVE-05
Requisito relacionado	RF-001 RF-003 RF-004 RF-005 RNF-001 RNF-002 RNF-003
Descripción	Cerrado de un fichero.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El proceso se ha conectado con la Base de Datos. 4. El fichero a introducir existe. 5. El fichero ha sido abierto por el proceso.
Procedimiento	1. El usuario llama a la función de cerrado del fichero. 2. El usuario introduce el identificador de fichero obtenido anteriormente.
Postcondiciones	1. Se reduce en uno el número de ficheros que tienen abierto el fichero en la Base de Datos. 2. El proceso elimina los datos que posee del fichero de la lista de ficheros abiertos.
Éxito	Sí.

TABLA 6.7. PRUEBA DE VERIFICACIÓN PVE-06

Identificador	PVE-06
Requisito relacionado	RF-001 RF-003 RF-004 RF-006 RNF-001 RNF-002 RNF-003
Descripción	Obtención metadatos del fichero.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El proceso se ha conectado con la Base de Datos. 4. El fichero a obtener metadatos existe. 5. El fichero ha sido abierto por el proceso.
Procedimiento	1. El usuario llama a la función de obtención de metadatos del fichero introduciendo su identificador.
Postcondiciones	1. Se obtienen los metadatos del fichero correctamente.
Éxito	Sí.

TABLA 6.8. PRUEBA DE VERIFICACIÓN PVE-07

Identificador	PVE-07
Requisito relacionado	RF-001 RF-003 RF-004 RF-007 RNF-001 RNF-002 RNF-003
Descripción	Modificación puntero del fichero.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El proceso se ha conectado con la Base de Datos. 4. El fichero a introducir existe. 5. El fichero ha sido abierto por el proceso.
Procedimiento	1. El usuario llama a la función de modificación del fichero. 2. El usuario introduce el identificador de fichero asociado cuyo puntero quiere modificar. 3. El usuario introduce la posición desde la que quiere empezar a mover el puntero. 4. El usuario introduce los bytes que quiere mover el puntero.
Postcondiciones	1. Se modifica el puntero correctamente.
Éxito	Sí.

TABLA 6.9. PRUEBA DE VERIFICACIÓN PVE-08

Identificador	PVE-08
Requisito relacionado	RF-001 RF-003 RF-006 RF-008 RNF-001 RNF-002 RNF-003
Descripción	Obtención metadatos de un directorio o de un fichero
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El proceso se ha conectado con la Base de Datos. 4. El elemento a obtener metadatos existe.
Procedimiento	1. El usuario llama a la función de obtención de metadatos introduciendo el path absoluto ya sea fichero o directorio.
Postcondiciones	1. Se obtienen los metadatos del elemento correctamente.
Éxito	Sí.

TABLA 6.10. PRUEBA DE VERIFICACIÓN PVE-09

Identificador	PVE-09
Requisito relacionado	RF-001 RF-003 RF-004 RF-007 RF-009 RNF-001 RNF-002 RNF-003
Descripción	Escritura de un fichero.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 4. El fichero a introducir existe. 5. El fichero ha sido abierto por el proceso.
Procedimiento	1. El usuario llama a la función de escritura de un fichero. 2. El usuario introduce el identificador del fichero que se quiere escribir. 3. El usuario introduce los datos que se quieren escribir. 4. El usuario introduce la cantidad de bytes que se quieren almacenar.
Postcondiciones	1. Se escriben los datos en el fichero correctamente.
Éxito	Sí.

TABLA 6.11. PRUEBA DE VERIFICACIÓN PVE-10

Identificador	PVE-10
Requisito relacionado	RF-001 RF-003 RF-004 RF-007 RF-010 RNF-001 RNF-002 RNF-003
Descripción	Lectura de un fichero.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 4. El fichero a leer existe. 5. El fichero ha sido abierto por el proceso.
Procedimiento	1. El usuario llama a la función de lectura de un fichero. 2. El usuario introduce el identificador del fichero que se quiere leer. 3. El usuario introduce el buffer donde guardar la información leída del fichero. 4. El usuario introduce la cantidad de bytes que se quieren leer.
Postcondiciones	1. Se leen los datos del fichero correctamente.
Éxito	Sí.

TABLA 6.12. PRUEBA DE VERIFICACIÓN PVE-11

Identificador	PVE-11
Requisito relacionado	RF-001 RF-011 RF-014 RNF-001 RNF-002 RNF-003
Descripción	Escritura de un directorio.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El directorio a abrir existe.
Procedimiento	1. El usuario llama a la función de apertura de un directorio. 2. Introduce el path absoluto del directorio a abrir.
Postcondiciones	1. Se abre el directorio correctamente. 2. Se guardan los datos del directorio en la lista de directorios abiertos del proceso.
Éxito	Sí.

TABLA 6.13. PRUEBA DE VERIFICACIÓN PVE-12

Identificador	PVE-12
Requisito relacionado	RF-001 RF-011 RF-012 RF-014 RNF-001 RNF-002 RNF-003
Descripción	Lectura de los elementos de un directorio.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El directorio a abrir existe. 4. El directorio se encuentra abierto en el proceso.
Procedimiento	1. El usuario llama a la función de lectura de un directorio. 2. Introduce el identificador del directorio abierto.
Postcondiciones	1. Se obtiene el siguiente elemento almacenado en el directorio.
Éxito	Sí.

TABLA 6.14. PRUEBA DE VERIFICACIÓN PVE-13

Identificador	PVE-13
Requisito relacionado	RF-001 RF-011 RF-013 RF-014 RNF-001 RNF-002 RNF-003
Descripción	Cerrar el directorio abierto.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El directorio a cerrar existe. 4. El directorio se encuentra abierto en el proceso.
Procedimiento	1. El usuario llama a la función de cerrado de un directorio. 2. Introduce el identificador del directorio abierto.
Postcondiciones	1. Se cierra el directorio. 2. Se eliminan los datos de la lista de directorios abiertos por el proceso.
Éxito	Sí.

TABLA 6.15. PRUEBA DE VERIFICACIÓN PVE-14

Identificador	PVE-14
Requisito relacionado	RF-001 RF-014 RNF-001 RNF-002 RNF-003
Descripción	Creación del directorio.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El directorio a crear no existe.
Procedimiento	1. El usuario llama a la función de creación de un directorio. 2. Introduce el path absoluto del directorio a crear. 3. Introduce los permisos de creación del directorio a crear.
Postcondiciones	1. Se crea el directorio correctamente en la Base de Datos.
Éxito	Sí.

TABLA 6.16. PRUEBA DE VERIFICACIÓN PVE-15

Identificador	PVE-15
Requisito relacionado	RF-001 RF-014 RF-015 RNF-001 RNF-002 RNF-003
Descripción	Borrado del directorio.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El directorio a borrar existe.
Procedimiento	1. El usuario llama a la función de borrado de un directorio. 2. Introduce el path absoluto del directorio a eliminar.
Postcondiciones	1. Se borra el directorio correctamente de la Base de Datos.
Éxito	Sí.

TABLA 6.17. PRUEBA DE VERIFICACIÓN PVE-16

Identificador	PVE-16
Requisito relacionado	RF-001 RF-003 RF-016 RNF-001 RNF-002 RNF-003
Descripción	Borrado del fichero.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos. 3. El fichero a borrar existe.
Procedimiento	1. El usuario llama a la función de borrado de un fichero. 2. Introduce el path absoluto del fichero a eliminar.
Postcondiciones	1. Se borra el fichero correctamente de la Base de Datos.
Éxito	Sí.

TABLA 6.18. PRUEBA DE VERIFICACIÓN PVE-17

Identificador	PVE-17
Requisito relacionado	RF-001 RF-002 RF-003 RF-004 RF-005 RF-006 RF-007 RF-008 RF-009 RF-010 RF-011 RF-012 RF-013 RF-014 RF-015 RF-016 RNF-001 RNF-002 RNF-003
Descripción	Manejo de errores.
Precondiciones	1. La Base de Datos se encuentra activa. 2. La interfaz posee las direcciones IP en las que se encuentra la Base de Datos.
Procedimiento	1. Se diseña una aplicación que llama a todas las funciones implementadas. 2. Se asegura que las llamadas a dichas funciones producen un fallo. 3. Se comprueba que el fallo devuelto por cada una de las funciones sigue el estándar POSIX.
Postcondiciones	1. Cada una de las funciones devuelve el código de error correcto.
Éxito	Sí.

TABLA 6.19. PRUEBA DE VERIFICACIÓN PVE-18

Identificador	PVE-18
Requisito relacionado	RF-001 RF-002 RF-003 RF-004 RF-005 RF-006 RF-007 RF-008 RF-009 RF-010 RF-011 RF-012 RF-013 RF-014 RF-015 RF-016 RNF-001 RNF-002 RNF-003
Descripción	El sistema está implementado en C.
Precondiciones	1. Se dispone de un compilador en C. 2. Se usa un entorno Linux.
Procedimiento	1. Se compila la nueva versión de la interfaz.
Postcondiciones	1. La interfaz se compila correctamente.
Éxito	Sí.

TABLA 6.20. PRUEBA DE VERIFICACIÓN PVE-19

Identificador	PVE-19
Requisito relacionado	RF-001 RF-002 RF-003 RF-004 RF-005 RF-009 RF-010 RF-017 RNF-001 RNF-002 RNF-003
Descripción	El sistema está integrado en MPI.
Precondiciones	1. La interfaz de Cassandra se encuentra en la configuración de MPI como otro Sistema de Ficheros. 2. Se usa un entorno Linux. 3. El cluster que conforma la Base de Datos está activo.
Procedimiento	1. Se abre un fichero. 2. Se escribe en un fichero. 3. Se lee de un fichero. 4. Se cierra un fichero.
Postcondiciones	1. Se han escrito los datos correctamente en el fichero. 2. Se han leído los datos del fichero correctamente.
Éxito	Sí.

Para finalizar y determinar que todos los requisitos definidos por el usuario se cumplen, se ha realizado la siguiente matriz de verificación que asocia cada prueba de verificación con los requisitos de usuario.

TABLA 6.21. MATRIZ DE PRUEBAS DE VERIFICACIÓN

-	PVE-01	PVE-02	PVE-03	PVE-04	PVE-05	PVE-06	PVE-07	PVE-08	PVE-09	PVE-10	PVE-11	PVE-12	PVE-13	PVE-14	PVE-15	PVE-16	PVE-17	PVE-18	PVE-19
RF-001	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RF-002	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓
RF-003	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	✓	✓	✓	✓
RF-004	✓	✓	-	✓	✓	✓	✓	-	✓	✓	-	-	-	-	-	-	✓	✓	✓
RF-005	✓	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓
RF-006	✓	✓	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	✓	✓	-
RF-007	✓	✓	-	-	-	-	✓	-	✓	✓	-	-	-	-	-	-	✓	✓	-
RF-008	✓	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	✓	-
RF-009	✓	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	✓	✓
RF-010	✓	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	✓	✓	✓
RF-011	✓	✓	-	-	-	-	-	-	-	-	✓	✓	✓	-	-	-	✓	✓	-
RF-012	✓	✓	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	✓	✓	-
RF-013	✓	✓	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	✓	✓	-
RF-014	✓	✓	-	-	-	-	-	✓	-	-	✓	✓	✓	✓	✓	-	✓	✓	-
RF-015	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	✓	✓	-
RF-016	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	-
RF-017	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓
RNF-001	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RNF-002	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RNF-003	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

6.1.2. Pruebas de validación

Después de desarrollar el sistema se debe determinar si su funcionamiento es el correcto y el que desea el usuario. Por eso, se van a definir varias pruebas de validación que determinarán si el proyecto cumple con lo establecido anteriormente. La plantilla que se va a usar para las pruebas de validación será la siguiente:

TABLA 6.22. PLANTILLA PARA LAS PRUEBAS DE VALIDACIÓN

Identificador	Sirve para identificar de forma unívoca la prueba. Seguirán la estructura PVA-XX, siendo PVA el acrónimo de Prueba de Verificación y XX serán dos dígitos.
Requisito relacionado	Identificador del requisito funcional o funcional que se verifica en la prueba que se realice.
Descripción	Explicación de la prueba de verificación que se está realizando.
Precondiciones	Condiciones previas a la realización de la prueba.
Procedimiento	Pasos a realizar para realizar la prueba.
Postcondiciones	Condición en la que se encuentra el sistema después de la prueba.
Éxito	Campo que indica si la prueba ha resultado satisfactoria o no.

Por tanto, las pruebas de validación que se han definido y realizado han sido las siguientes:

TABLA 6.23. PRUEBA DE VALIDACIÓN PVA-01

Identificador	PVA-01
Requisito relacionado	RF-001 RF-002 RF-003 RF-004 RF-005 RF-017 RNF-001 RNF-002 RNF-003
Descripción	Apertura y cerrado un fichero de la Base de Datos Apache Cassandra a través de llamadas MPI.
Precondiciones	1. La interfaz de Cassandra se encuentra en la configuración de MPI como otro Sistema de Ficheros. 2. Se usa un entorno Linux. 3. El cluster que conforma la Base de Datos está activo.
Procedimiento	1. La aplicación llama a MPI_Init. 2. La aplicación llama a la función MPI_File_Open para abrir el fichero indicando la ruta en la que se encuentra el fichero e indicando el Sistema de Ficheros que se va a usar (cassandra:). La apertura se realiza en modo de lectura y si no existe también se crea. 3. La aplicación llama a la función MPI_File_Close para cerrar el fichero. 4. La aplicación llama a la función MPI_Finalize y acaba la ejecución
Postcondiciones	El proceso abre y cierra el fichero sin error alguno.
Éxito	Sí.

TABLA 6.24. PRUEBA DE VALIDACIÓN PVA-02

Identificador	PVA-02
Requisito relacionado	RF-001 RF-002 RF-003 RF-004 RF-005 RF-010 RF-017 RNF-001 RNF-002 RNF-003
Descripción	Lectura de un fichero de la Base de Datos Apache Cassandra a través de llamadas MPI.
Precondiciones	1. La interfaz de Cassandra se encuentra en la configuración de MPI como otro Sistema de Ficheros. 2. Se usa un entorno Linux. 3. El cluster que conforma la Base de Datos está activo. 4. Existe un fichero en la Base de Datos.
Procedimiento	1. La aplicación llama a MPI_Init. 2. La aplicación llama a la función MPI_File_Open para abrir el fichero indicando la ruta en la que se encuentra el fichero e indicando el Sistema de Ficheros que se va a usar (cassandra:). La apertura se realiza en modo de lectura. 3. La aplicación llama a la función MPI_File_Read para leer los datos guardados en el fichero. 4. La aplicación llama a la función MPI_File_Close para cerrar el fichero. 5. La aplicación llama a la función MPI_Finalize y acaba la ejecución.
Postcondiciones	El proceso lee los datos del fichero sin error alguno.
Éxito	Sí.

TABLA 6.25. PRUEBA DE VALIDACIÓN PVA-03

Identificador	PVA-03
Requisito relacionado	RF-001 RF-002 RF-004 RF-005 RF-009 RF-017 RNF-001 RNF-002 RNF-003
Descripción	Escritura en un fichero de la Base de Datos Apache Cassandra a través de llamadas MPI.
Precondiciones	1. La interfaz de Cassandra se encuentra en la configuración de MPI como otro Sistema de Ficheros. 2. Se usa un entorno Linux. 3. El cluster que conforma la Base de Datos está activo. 4. Existe un fichero en la Base de Datos.
Procedimiento	1. La aplicación llama a MPI_Init. 2. La aplicación llama a la función MPI_File_Open para abrir el fichero indicando la ruta en la que se encuentra el fichero e indicando el Sistema de Ficheros que se va a usar (cassandra:). La apertura se realiza en modo de lectura. 3. La aplicación llama a la función MPI_File_Write para escribir los datos en el fichero. 4. La aplicación llama a la función MPI_File_Close para cerrar el fichero. 5. La aplicación llama a la función MPI_Finalize y acaba la ejecución.
Postcondiciones	El proceso escribe los datos del fichero sin error alguno.
Éxito	Sí.

TABLA 6.26. PRUEBA DE VALIDACIÓN PVA-04

Identificador	PVA-04
Requisito relacionado	RF-001 RF-002 RF-004 RF-005 RF-010 RNF-001 RNF-002 RNF-003
Descripción	Lectura de un fichero de gran tamaño por varios clientes de la Base de Datos.
Precondiciones	1. Se usa un entorno Linux. 2. El cluster que conforma la Base de Datos está activo. 3. Existe un fichero en la Base de Datos.
Procedimiento	1. Se indica a la aplicación el número de procesos participantes. 2. Se calcula la posición por la que empieza cada proceso a leer. 3. Se calcula el tamaño de datos que cada proceso tiene que leer. 4. Se abre el fichero de la Base de Datos. 5. Se escriben los datos de forma local en un fichero. 6. Se cierra el fichero de la Base de Datos.
Postcondiciones	El proceso escribe los datos del fichero de forma local sin error alguno.
Éxito	Sí.

TABLA 6.27. PRUEBA DE VALIDACIÓN PVA-05

Identificador	PVA-05
Requisito relacionado	RF-001 RF-002 RF-004 RF-005 RF-009 RNF-001 RNF-002 RNF-003
Descripción	Escritura de un fichero de gran tamaño por varios clientes de la Base de Datos.
Precondiciones	1. Se usa un entorno Linux. 2. El cluster que conforma la Base de Datos está activo. 3. Existe un fichero en la Base de Datos.
Procedimiento	1. Se indica a la aplicación el número de procesos participantes. 2. Se calcula la posición por la que empieza cada proceso a escribir. 3. Se calcula el tamaño de datos que cada proceso tiene que escribir. 4. Se abre el fichero de la Base de Datos. 5. Se leen los datos del fichero local y se escriben en la Base de Datos. 6. Se cierra el fichero de la Base de Datos.
Postcondiciones	El proceso escribe los datos en la Base de Datos sin error alguno.
Éxito	Sí.

TABLA 6.28. PRUEBA DE VALIDACIÓN PVA-06

Identificador	PVA-06
Requisito relacionado	RF-001 RF-002 RF-004 RF-005 RF-007 RNF-001 RNF-002 RNF-003
Descripción	Modificación del puntero del fichero local.
Precondiciones	1. Se usa un entorno Linux. 2. El cluster que conforma la Base de Datos está activo. 3. Existe un fichero en la Base de Datos.
Procedimiento	1. Se abre el fichero de la Base de Datos. 2. Se modifica el puntero a la posición deseada. 3. Se escriben o leen los datos de la Base de Datos. 4. Se cierra el fichero de la Base de Datos.
Postcondiciones	El proceso realiza las operaciones sin error alguno.
Éxito	Sí.

TABLA 6.29. PRUEBA DE VALIDACIÓN PVA-07

Identificador	PVA-07
Requisito relacionado	RF-001 RF-002 RF-011 RF-012 RF-013 RF-014 RF-015 RNF-001 RNF-002 RNF-003
Descripción	Manipulación de directorios por un cliente.
Precondiciones	1. Se usa un entorno Linux. 2. El cluster que conforma la Base de Datos está activo. 3. Existe al menos un directorio en la Base de Datos.
Procedimiento	1. Se abre un directorio existente de la Base de Datos. 2. Se crea un nuevo directorio. 3. Se borra otro directorio. 4. Se leen los elementos del directorio abierto. 5. Se cierra el directorio abierto de la Base de Datos.
Postcondiciones	El proceso realiza las operaciones sin error alguno.
Éxito	Sí.

TABLA 6.30. PRUEBA DE VALIDACIÓN PVA-08

Identificador	PVA-08
Requisito relacionado	RF-001 RF-002 RF-003 RF-004 RF-005 RF-007 RF-009 RF-010 RF-016 RNF-001 RNF-002 RNF-003
Descripción	Manipulación de ficheros por un cliente.
Precondiciones	1. Se usa un entorno Linux. 2. El cluster que conforma la Base de Datos está activo. 3. Existe al menos un directorio en la Base de Datos.
Procedimiento	1. Se abre un fichero existente de la Base de Datos. 2. Se crea un nuevo fichero. 3. Se borra otro fichero. 4. Se leen los elementos del fichero abierto. 5. Se cierra el fichero abierto de la Base de Datos.
Postcondiciones	El proceso realiza las operaciones sin error alguno.
Éxito	Sí.

TABLA 6.31. PRUEBA DE VALIDACIÓN PVA-09

Identificador	PVA-09
Requisito relacionado	RF-001 RF-002 RF-006 RF-008 RNF-001 RNF-002 RNF-003
Descripción	Obtención metadatos.
Precondiciones	1. Se usa un entorno Linux. 2. El cluster que conforma la Base de Datos está activo. 3. Existe al menos un directorio y un fichero en la Base de Datos.
Procedimiento	1. Se obtienen los metadatos de un fichero existente en la Base de Datos. 2. Se obtienen los metadatos de un directorio existente en la Base de Datos.
Postcondiciones	El proceso realiza las operaciones sin error alguno.
Éxito	Sí.

Para finalizar y determinar que todos el sistema funciona correctamente tal y como desea el usuario, se ha realizado la siguiente matriz de validación que asocia cada prueba de validación con los requisitos.

TABLA 6.32. MATRIZ DE PRUEBAS DE VERIFICACIÓN

-	PVA-01	PVA-02	PVA-03	PVA-04	PVA-05	PVA-06	PVA-07	PVA-08	PVA-09
RF-001	✓	✓	✓	✓	✓	✓	✓	✓	✓
RF-002	✓	✓	✓	✓	✓	✓	✓	✓	✓
RF-003	✓	✓	-	-	-	-	-	✓	-
RF-004	✓	✓	✓	✓	✓	✓	-	✓	-
RF-005	✓	✓	✓	✓	✓	✓	-	✓	-
RF-006	-	-	-	-	-	-	-	-	✓
RF-007	-	-	-	-	-	✓	-	✓	-
RF-008	-	-	-	-	-	-	-	-	✓
RF-009	-	-	✓	-	✓	-	-	✓	-
RF-010	-	✓	-	✓	-	-	-	✓	-
RF-011	-	-	-	-	-	-	✓	-	-
RF-012	-	-	-	-	-	-	✓	-	-
RF-013	-	-	-	-	-	-	✓	-	-
RF-014	-	-	-	-	-	-	✓	-	-
RF-015	-	-	-	-	-	-	✓	-	-
RF-016	-	-	-	-	-	-	-	✓	-
RF-017	✓	✓	✓	-	-	-	-	-	-
RNF-001	✓	✓	✓	✓	✓	✓	✓	✓	✓
RNF-002	✓	✓	✓	✓	✓	✓	✓	✓	✓
RNF-003	✓	✓	✓	✓	✓	✓	✓	✓	✓

6.2 Rendimiento de la interfaz

En esta sección se va a estudiar el rendimiento de la interfaz desarrollada para acceder a Apache Cassandra. En primer lugar se mostrarán las pruebas que se han realizado sobre Cassandra utilizando diferentes tamaños de fichero y de tamaño de bloque con diferentes máquinas y, después, se mostrará otra comparativa de los rendimientos que ha mostrado

Cassandra con los resultados que ha mostrado el Sistema de Ficheros HDFS [1].

Para las pruebas de Apache Cassandra se han utilizado un cluster de 4 y de 8 máquinas, mientras que, para la segunda comparativa, se ha utilizado únicamente un cluster de 8 máquinas para asemejarse a la configuración que tiene HDFS en el cluster.

Cada una de estas máquinas dispone de un procesador Intel Xeon E405, con 8GB de RAM y con un almacenamiento en disco de 1TB de datos. En cada una de estas máquinas se ha realizado la instalación de Apache Cassandra con la misma configuración. Además, cabe destacar que tanto para la Base de Datos como para el Sistema de Ficheros se ha eliminado la información de las cachés tanto para lectura como para escritura, para que ambos sistemas se encuentren en igualdad de condiciones.

Todas las pruebas que se van a realizar en ambos sistemas se van a utilizar los mismos ficheros con los siguientes tamaños: 1MiB, 512MiB y 1024MiB. Además, se ha estado cambiando el tamaño de bloque y el factor de replicación (Factor de replicación 1 y 3) para comprobar en qué casos ambos sistemas pueden hacer las operaciones más rápidamente. A continuación, se van a mostrar las arquitecturas utilizadas para el cluster de Apache Cassandra y para el cluster de HDFS.

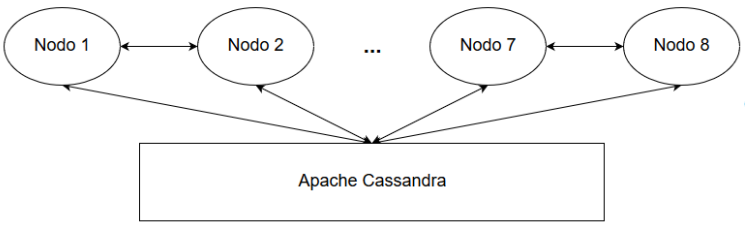


Fig. 6.1. Arquitectura del cluster de Cassandra para pruebas

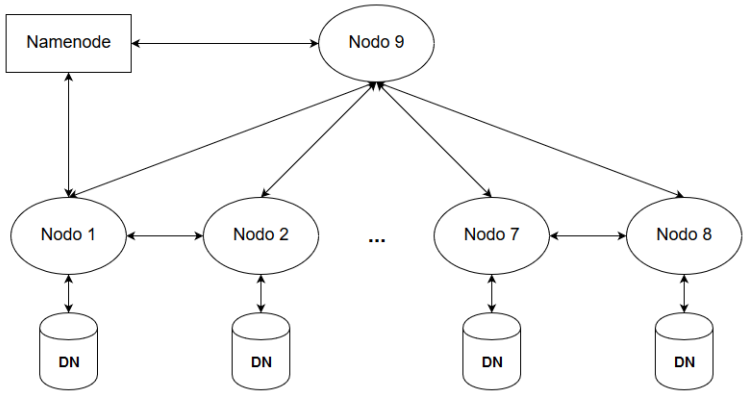


Fig. 6.2. Arquitectura del cluster de HDFS para pruebas

6.2.1. Comparativa de tasas de transferencia de la interfaz de Cassandra

Lo que se quiere conseguir con estas pruebas es determinar en qué casos se comporta mejor la Base de Datos recibiendo mayor o menor carga de datos modificando el bloque de datos que se le envía desde el usuario a la hora de escribir información de un fichero o la información que recibe el usuario si quiere leer datos guardados en Cassandra. En primer lugar, se han realizado pruebas de escritura y lectura no paralelas, es decir, solo hay un único proceso a la vez que accede al fichero. Estas pruebas se han realizado sobre los ficheros anteriormente mencionados con tamaños de bloque de 32KiB, 64KiB, 128KiB, 1MiB, 32MiB, 64MiB y 128MiB, tamaños iguales al tamaño de acceso a los ficheros de manera local por el proceso.

Escritura no paralela

En estas pruebas, se tiene un único proceso escribiendo en la Base de Datos de forma secuencial. En el caso de Apache Cassandra, no existe localidad, es decir, da igual desde qué nodo del cluster realicemos la escritura y lectura de los datos, puesto que Cassandra distribuirá la información entre los nodos según desee, por lo que todas las pruebas se realizaron en el nodo 1. Los resultados obtenidos son los siguientes:

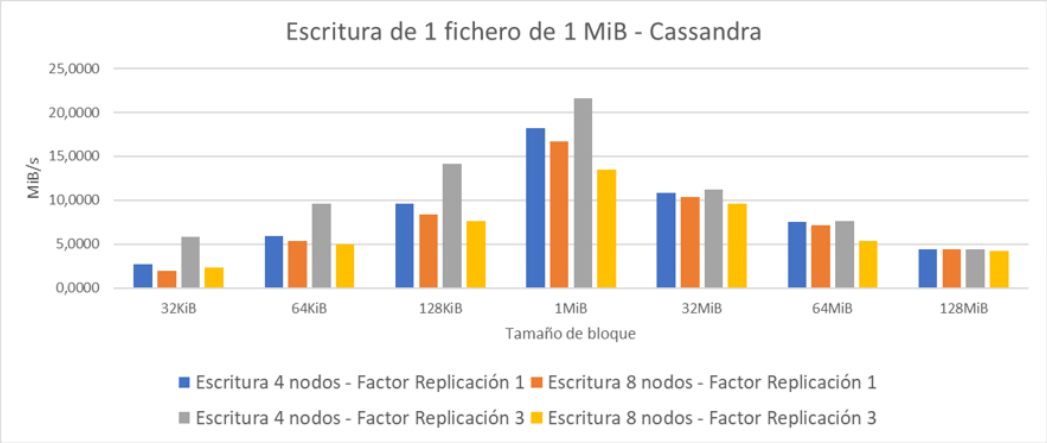


Fig. 6.3. Escritura de 1 fichero de 1 MiB

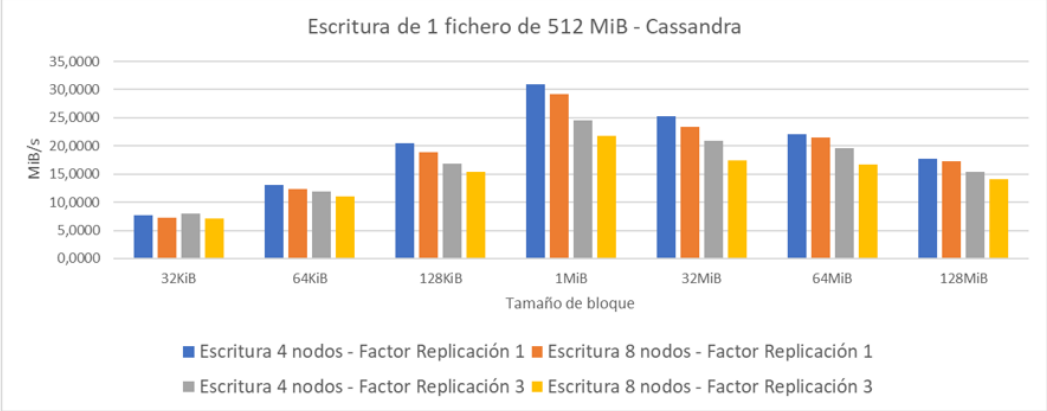


Fig. 6.4. Escritura de 1 fichero de 512 MiB

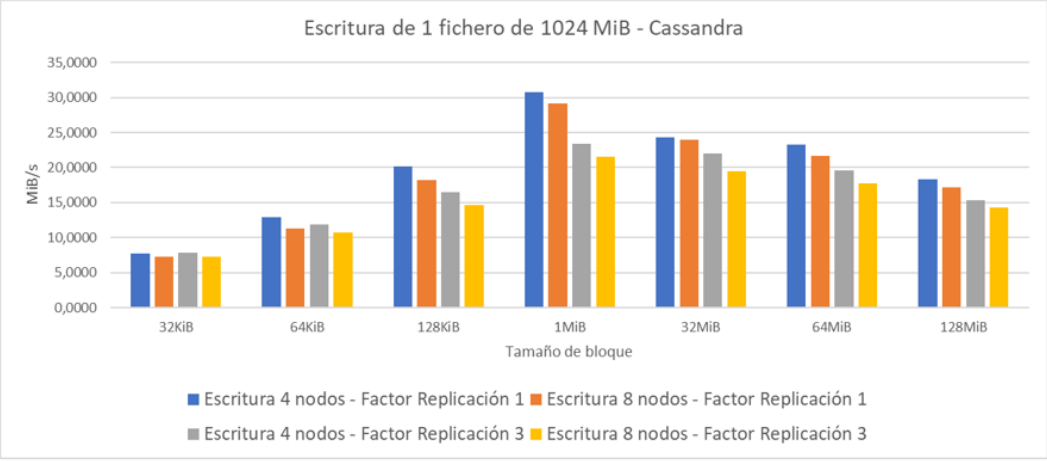


Fig. 6.5. Escritura de 1 fichero de 1024 MiB

Como se puede apreciar en las figuras 6.3 6.4 y 6.5, el tamaño de bloque que mejores tasas de transferencia ofrece es el de 1MiB. A partir de este tamaño, ya sean más pequeños o más grandes, las tasas de transferencias se ven reducidas. Otro detalle importante a destacar es que, en la mayoría de casos, la escritura en 4 nodos y con un factor de replicación de 1, las tasas son mejores que las del resto, detalle que se debe tener muy en cuenta porque, en caso de fallo, no habría otra copia de los datos y puede darse el caso de que se corrompa o se pierda, no obstante, al solo estar replicado en un único nodo, se obtiene a una mayor velocidad.

Lectura no paralela

De igual forma que en el subapartado anterior, se tiene un único proceso leyendo de forma secuencia de Apache Cassandra. Los resultados obtenidos han sido los siguientes:

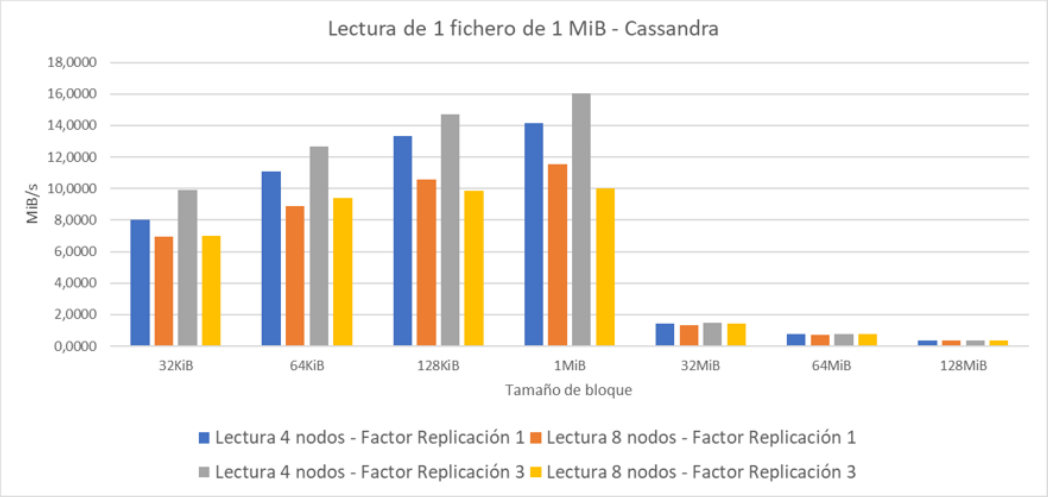


Fig. 6.6. Lectura de 1 fichero de 1 MiB

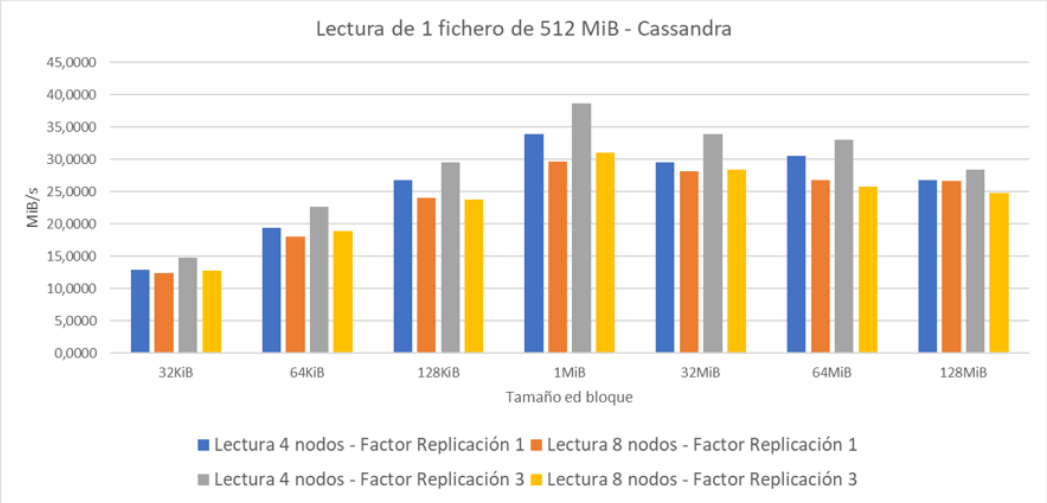


Fig. 6.7. Lectura de 1 fichero de 512 MiB

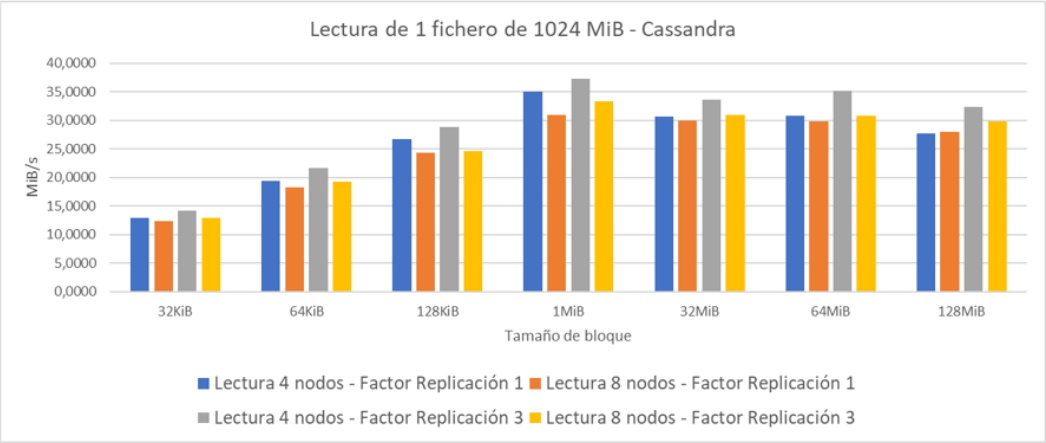


Fig. 6.8. Lectura de 1 fichero de 1024 MiB

Como se puede ver en la figura 6.6, con tamaños menores a 32MiB la tasa de transferencia de un fichero de 1MiB es hasta 8 veces mayor que si el tamaño de bloque elegido es de 32MiB, 64MiB y 128MiB y esto se debe a que, al tratarse de un fichero de 1MiB, existe fragmentación interna, por lo que el sistema perderá mucho tiempo leyendo datos basura innecesarios para el fichero. En cambio, si el fichero es de 512MiB y 1024MiB, como se ve en las figuras 6.7 y 6.8, las tasas de transferencia van mejorando hasta que el tamaño de bloque es de 1MiB, siendo la lectura con 4 nodos y factor de replicación 3 la opción que mejor velocidad ofrece con respecto a las otras opciones que en las figuras se muestran. También cabe destacar que para tamaños de bloque mayores a 1MiB, las tasas se ven reducidas ligeramente. En estos casos, habría que elegir si se prefiere un menor

tamaño de bloque a cambio de una mayor cantidad de peticiones a la Base de Datos, lo que implicaría una mayor sobrecarga de peticiones que recibiría Cassandra en el caso de que muchos usuarios lo utilizasen a la vez el Sistema.

6.2.2. Comparativa de tasas de transferencia entre Cassandra y HDFS

En este apartado se mostrará la comparativa de las velocidades de transferencia de datos que ofrecen tanto la Base de Datos usada para este proyecto como HDFS. Para ello, se van a mostrar las pruebas de lectura y escritura que se han hecho con un único cliente, es decir, estas ejecuciones acceden a la información de forma secuencial. Y, por último, se mostrará una comparativa más general mostrando las tasas de transferencia de escritura y lectura con un tamaño de acceso al fichero diferente al tamaño de bloque, que será de 64MiB y se incluirá tanto lectura y escritura secuencial y paralela. No obstante, hay que destacar que HDFS no ofrece la posibilidad de escribir en un mismo fichero por varios procesos a la vez, por lo que no se tendrá en cuenta.

Escritura no paralela

Tanto en este subapartado como en el siguiente de lectura, se va a proceder a realizar operaciones de escritura y lectura, respectivamente, de forma secuencial. Como se puede comprobar, para estas pruebas, se ha utilizado el cluster de 8 nodos y, además, los tamaños de bloque usados serán de 1MiB, 32MiB, 64MiB y 128MiB.

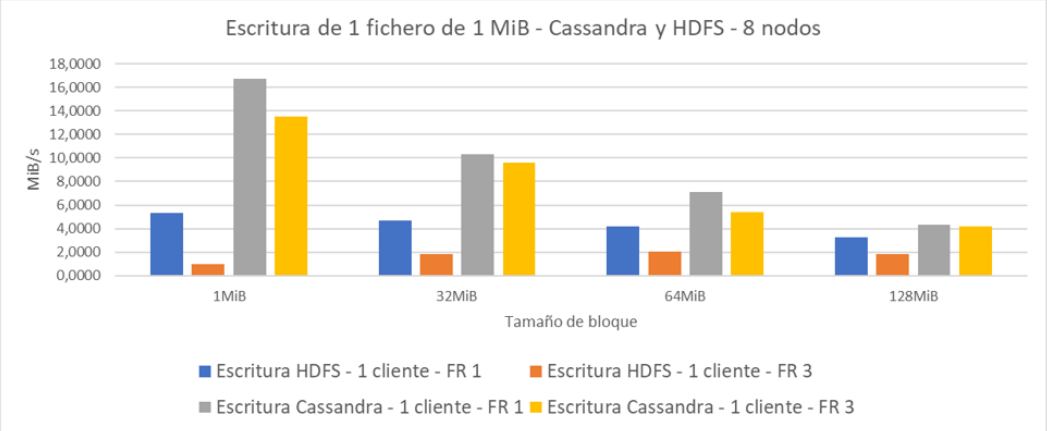


Fig. 6.9. Escritura de 1 fichero de 1 MiB Cassandra y HDFS

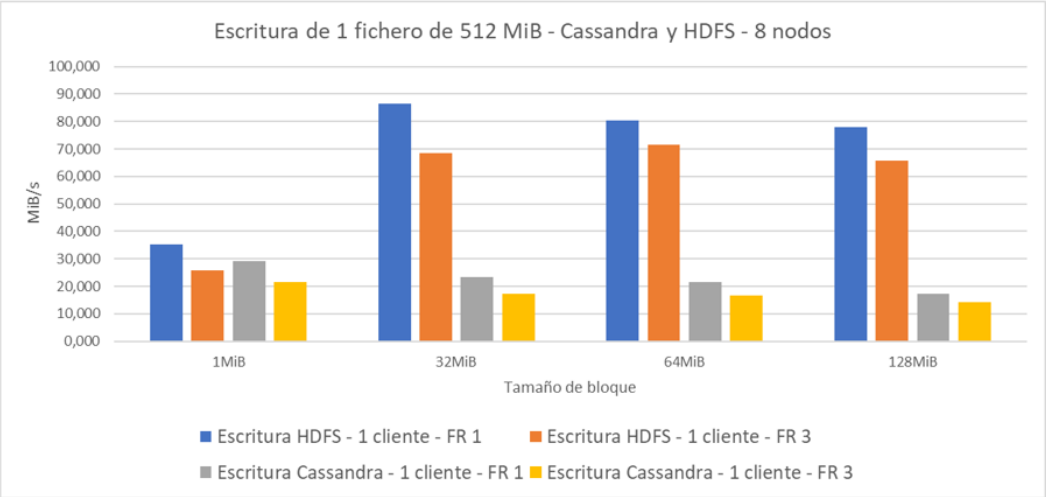


Fig. 6.10. Escritura de 1 fichero de 512 MiB Cassandra y HDFS

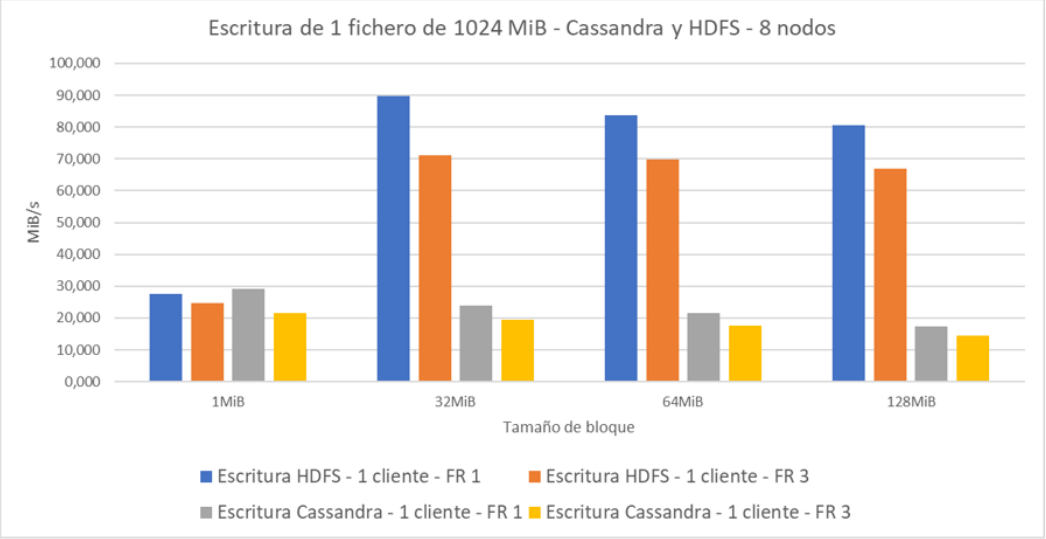


Fig. 6.11. Escritura de 1 fichero de 1024 MiB Cassandra y HDFS

Como se puede ver en la figura 6.9, HDFS presenta peores resultados con respecto a la Base de Datos cuando se quiere escribir un fichero de poco tamaño, como ocurre con el fichero de 1MiB. No obstante, se puede comprobar que, en las figuras 6.10 y 6.11, la tasa de transferencia de HDFS mejora considerablemente llegando a ser, aproximadamente, hasta 5 veces más rápido que las tasas de transferencia de Cassandra.

Lectura no paralela

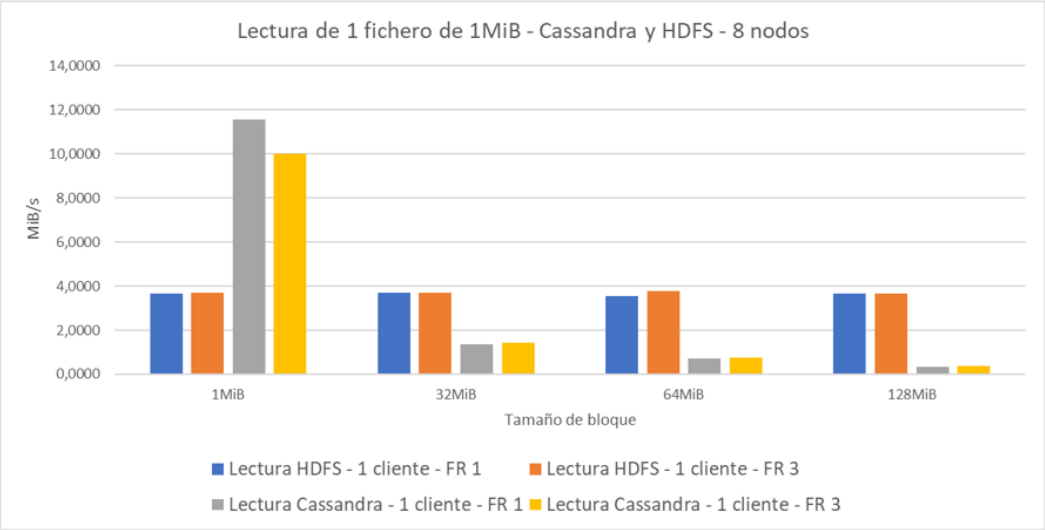


Fig. 6.12. Lectura de 1 fichero de 1 MiB Cassandra y HDFS

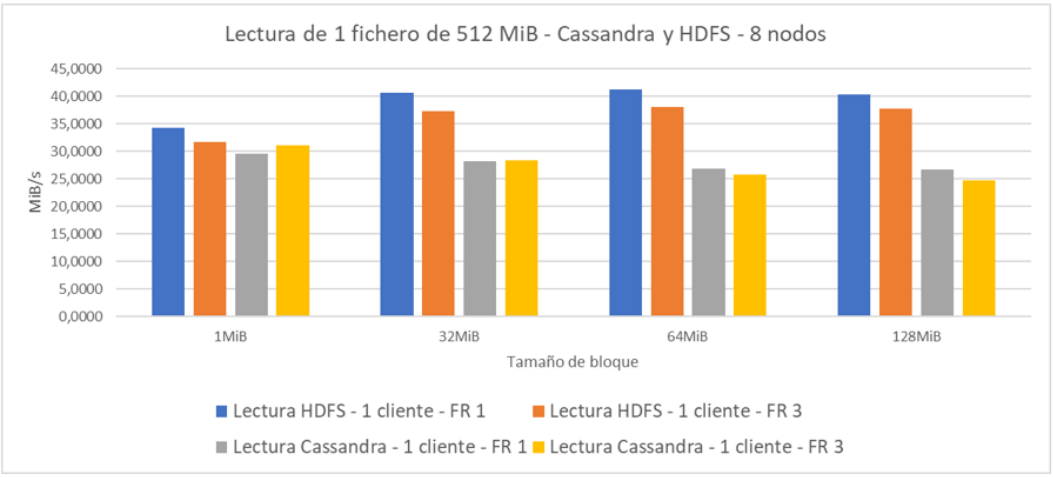


Fig. 6.13. Lectura de 1 fichero de 512 MiB Cassandra y HDFS

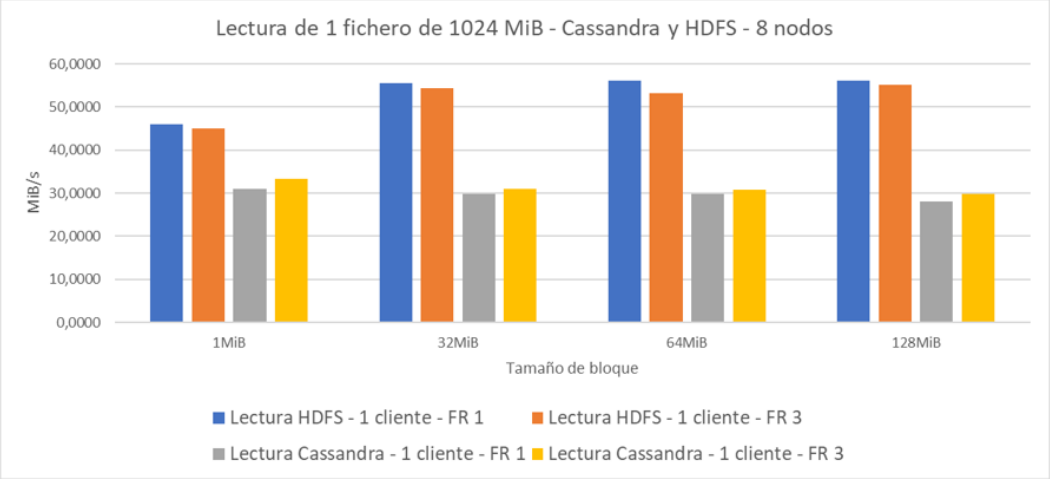


Fig. 6.14. Lectura de 1 fichero de 1024 MiB Cassandra y HDFS

De forma similar a la escritura, HDFS presenta mejores tasas de transferencia en comparación con los resultados de Cassandra, llegando a doblar las velocidades en algunos casos. Por otro lado, Cassandra ofrece unos resultados similares, como se pueden ver en las figuras 6.13 y 6.14, tanto con factor de replicación 1 como con el factor 3. No obstante, en la figura 6.12, de igual forma que ocurrió con la escritura no paralela, Cassandra muestra mejores resultados con un fichero de 1MiB, aunque con tamaño de bloque mayor a 1MiB vuelve a ocurrir la fragmentación interna y empeora la tasa de transferencia.

Escritura y lectura paralela y no paralela

A continuación, se van a mostrar los datos obtenidos con respecto a las escrituras y lecturas que se han realizado con diferentes tamaños de acceso a los ficheros, pero con un tamaño de bloque de 64 MiB. Además, se compararán estos resultados entre HDFS y Cassandra con un factor de replicación de 3 tanto para un proceso como para varios procesos que accedan al fichero de forma paralela. Cabe destacar que HDFS no ofrece la posibilidad de escritura paralela, pero se mostrarán las tasas de transferencia que realiza con 1 cliente.

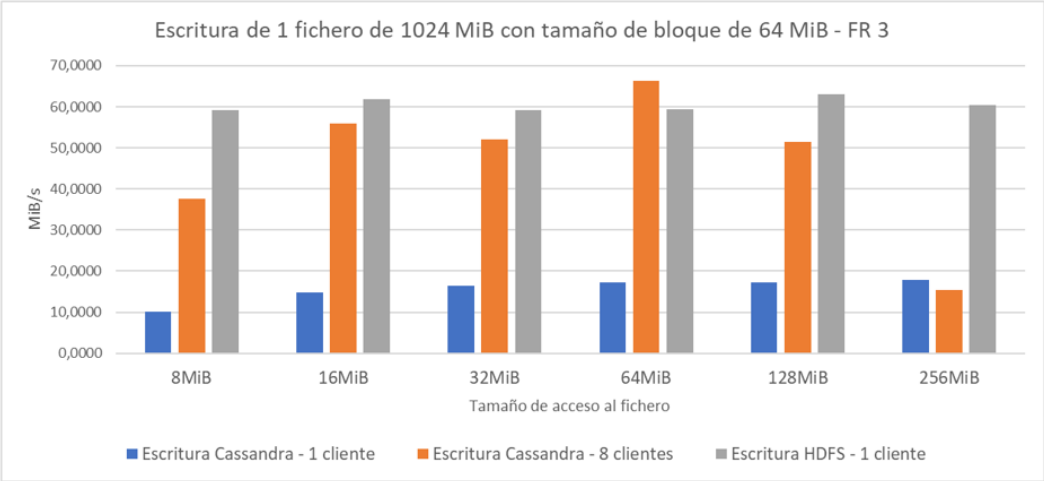


Fig. 6.15. Escritura de un fichero 1024 MiB con tamaño de acceso diferente

Como se puede ver en la figura 6.15, Cassandra mejora hasta 5 o 6 veces la velocidad de transferencia si se escribe un fichero por varios procesos que si lo escribiese solo uno. En cambio, a medida que aumenta el tamaño de acceso al fichero por encima del tamaño de bloque, las tasas empeoran drásticamente. Por otro lado, HDFS se mantiene por encima de los 50 MiB/s para todas las pruebas y Cassandra supera la mayor velocidad de HDFS si el tamaño de acceso al fichero coincide con el del tamaño de bloque.

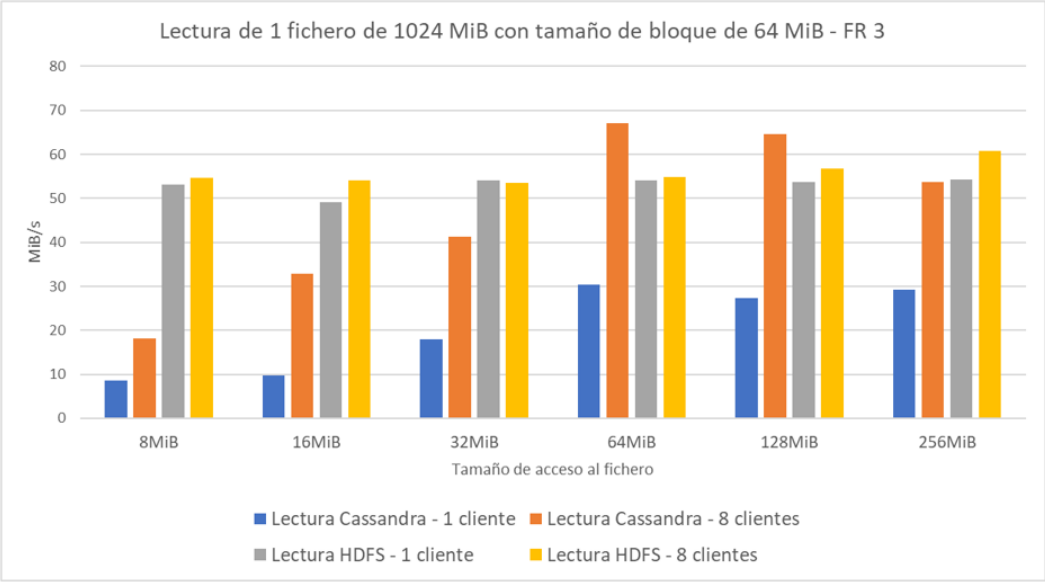


Fig. 6.16. Lectura de un fichero 1024 MiB con tamaño de acceso diferente

Por último, con respecto a las lecturas, Cassandra supera las velocidades de HDFS si se accede a los ficheros con tamaños de bloque de 64MiB y 128MiB, mientras que, para el resto de casos, HDFS muestra unas tasas de transferencia casi constantes.

6.2.3. Comparativa de tasas de transferencia entre Cassandra y MPI-IO

Para terminar, se van a mostrar las tasas de transferencia que ofrece Cassandra con las tasas que ofrece la interfaz desarrollada que ha sido integrada en MPI-IO. Lo que se quiere conseguir con esta comparativa es determinar la viabilidad de esta integración, comprobando que no supone mucha sobrecarga a la interfaz si se integrase de forma completa. Para la comparativa, se utilizará un fichero de 1GiB y se utilizarán diferentes tamaños de acceso al fichero que son 16MiB, 32MiB, 64MiB y 128MiB.

Escritura y lectura

Para estas últimas pruebas, se van a obtener las tasas de transferencia que ofrece la interfaz integrada en MPI-IO con varios procesos, comparándolo con las tasas que ofrece si no se integrase, tanto con varios procesos escribiendo o leyendo, como con un único proceso. Con un factor de replicación 3 y un tamaño de bloque de 64MiB, las tasas de escritura y lectura son:

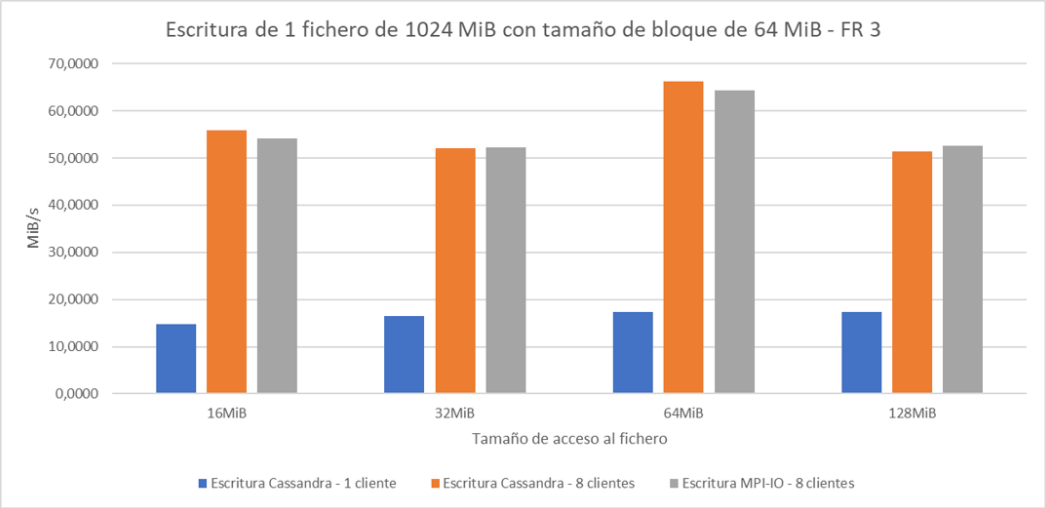


Fig. 6.17. Escritura de un fichero 1024 MiB con tamaño de acceso diferente Cassandra y MPI

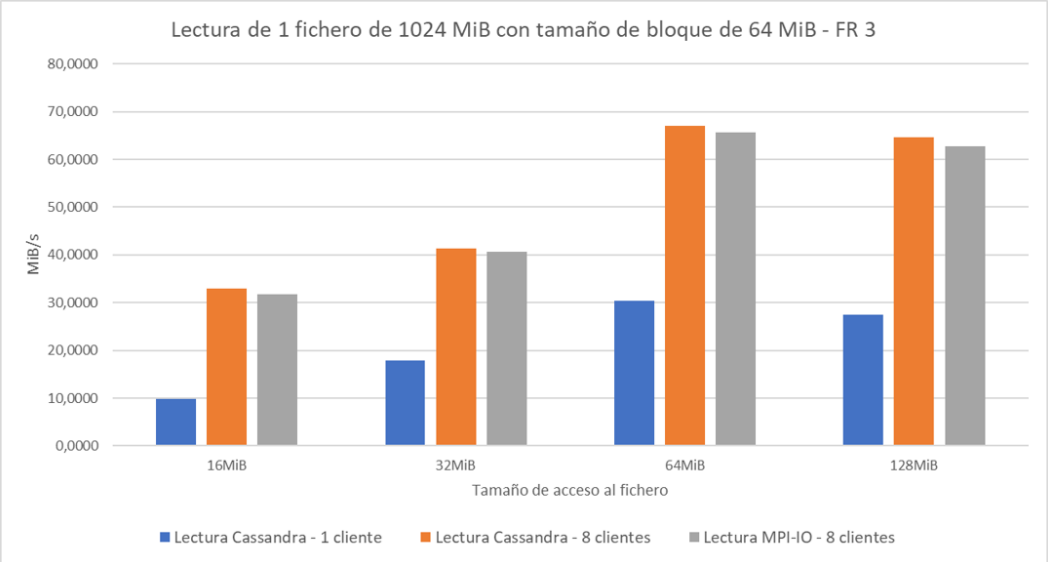


Fig. 6.18. Lectura de un fichero 1024 MiB con tamaño de acceso diferente Cassandra y MPI

Como se puede ver, tanto en la escritura como en la lectura, tal y como se muestra en las figuras 6.17 y 6.18, el uso de MPI-IO no supone una sobrecarga extra que empeore las tasas de transferencia en comparación con las tasas que se obtienen si no se integrase en MPI-IO. Por lo tanto, se puede decir que es posible la integración de la interfaz sin repercutir en el rendimiento del sistema.

Capítulo 7

Planificación y costes

En este capítulo, en primer lugar se va a mostrar la planificación que se ha seguido durante la realización de este proyecto (ver sección 7.1), seguido del análisis de los costes que ha conllevado con su consiguiente presupuesto (ver sección 7.2) y, para finalizar, se va a detallar el entorno socio-económico (ver sección 7.3) de este proyecto.

7.1 Planificación

En esta sección se va a mostrar la planificación que se ha realizado durante este TFG. A lo largo de esta parte se van a tener en cuenta los dos objetivos más importantes a conseguir que se definen en la sección 1.2 de la Introducción de esta memoria, que son:

- Interfaz de Cassandra basada en POSIX.
- Pruebas de rendimiento de la interfaz y de HDFS.

Para poder estructurar y organizar este proyecto de manera correcta, se tuvo que seleccionar una metodología de desarrollo de entre los tres modelos siguientes: modelo en cascada, modelo de prototipos y modelo en espiral. El modelo de desarrollo en cascada [46] tiene un enfoque secuencial para el desarrollo de proyectos software separándolo en varias fases, empezando con la especificación de los requisitos que establece el cliente, seguido del diseño del sistema, de la implementación, de la verificación y, por último del mantenimiento y despliegue.

El siguiente modelo de desarrollo a analizar, el modelo de prototipos [46] [47], tipo de modelo relacionado con el desarrollo evolutivo. En el caso del proyecto que se describe

en esta memoria, este modelo es el menos adecuado, porque el modelo de prototipos se caracteriza en que el propio prototipo debe ser construido en poco tiempo y que, además, puede ser modificado conforme avanza su desarrollo, de tal forma que estas características no se pueden aplicar en este TFG debido a que requería un análisis más profundo.

Por último, el modelo en espiral [46][48] es el modelo que más se adecua a este proyecto puesto que permite separar el desarrollo del sistema en segmentos y, además, se adapta conforme avanza el proyecto, permitiendo una mayor flexibilidad en comparación con los anteriores. Esto permite que, al dividir en diferentes fases el desarrollo y poder repetirlas una y otra vez, se puedan pulir los elementos que conforman el proyecto hasta que haya sido completado. En este caso, se va a usar una variante de este modelo de 4 regiones en el que se van a añadir 2 más (modelo de 6 regiones), variante que se adapta durante el desarrollo del sistema software. Este modelo está formado por los siguientes componentes:

- Comunicación con el cliente: Obtención de los requisitos que desea el cliente.
- Planificación: Fase en la que se especifican los objetivos, las opciones disponibles y las limitaciones.
- Análisis: Fase en la que se contemplan y analizan las distintas opciones y los riesgos de cada una.
- Ingeniería: Etapa de diseño del sistema a desarrollar.
- Desarrollo y pruebas: Fase de implementación del sistema pedido por el cliente y sus consiguientes comprobaciones.
- Evaluación del cliente: Fase de valoración del sistema por parte del cliente.

Una vez que se tienen las diferentes fases que posee cada iteración, se procede a explicar las iteraciones por las que ha pasado el sistema hasta que se ha completado. Como se puede observar en la siguiente lista, hay dos bloques diferenciados, el primer bloque consta de las 3 primeras iteraciones donde se desarrolla la interfaz de Apache Cassandra, mientras que las dos últimas iteraciones se corresponden con las ejecuciones de las funciones que se han diseñado en el proyecto para comprobar el rendimiento con

Apache Cassandra junto con el uso de las funciones que ofrece HDFS para determinar, por otro lado, su rendimiento y poder compararlas después.

- Iteración 1: Funciones de manipulación de ficheros en la Base de Datos.
- Iteración 2: Funciones de manipulación de directorios en la Base de Datos.
- Iteración 3: Funciones de obtención de metadatos de ficheros y directorios.
- Iteración 4: Ejecución de las funciones de manipulación de ficheros en la Base de Datos.
- Iteración 5: Ejecución de las funciones de manipulación de ficheros de HDFS.

7.1.1. Tiempo estimado

El tiempo que se ha establecido para poder completar este proyecto ha sido de 9 meses, más concretamente, la duración del proyecto se sitúa entre el 1 de Septiembre de 2018 y el 1 de Junio de 2019.

Para poder mostrar la evolución del proyecto durante el período estimado, se va a utilizar un diagrama de Gantt que presenta el tiempo de vida del proyecto al que se le ha añadido una etapa extra, llamada “Documentación”, la cual muestra el progreso de documentación de la evolución del TFG. El diagrama de Gantt que se ha obtenido es el siguiente:

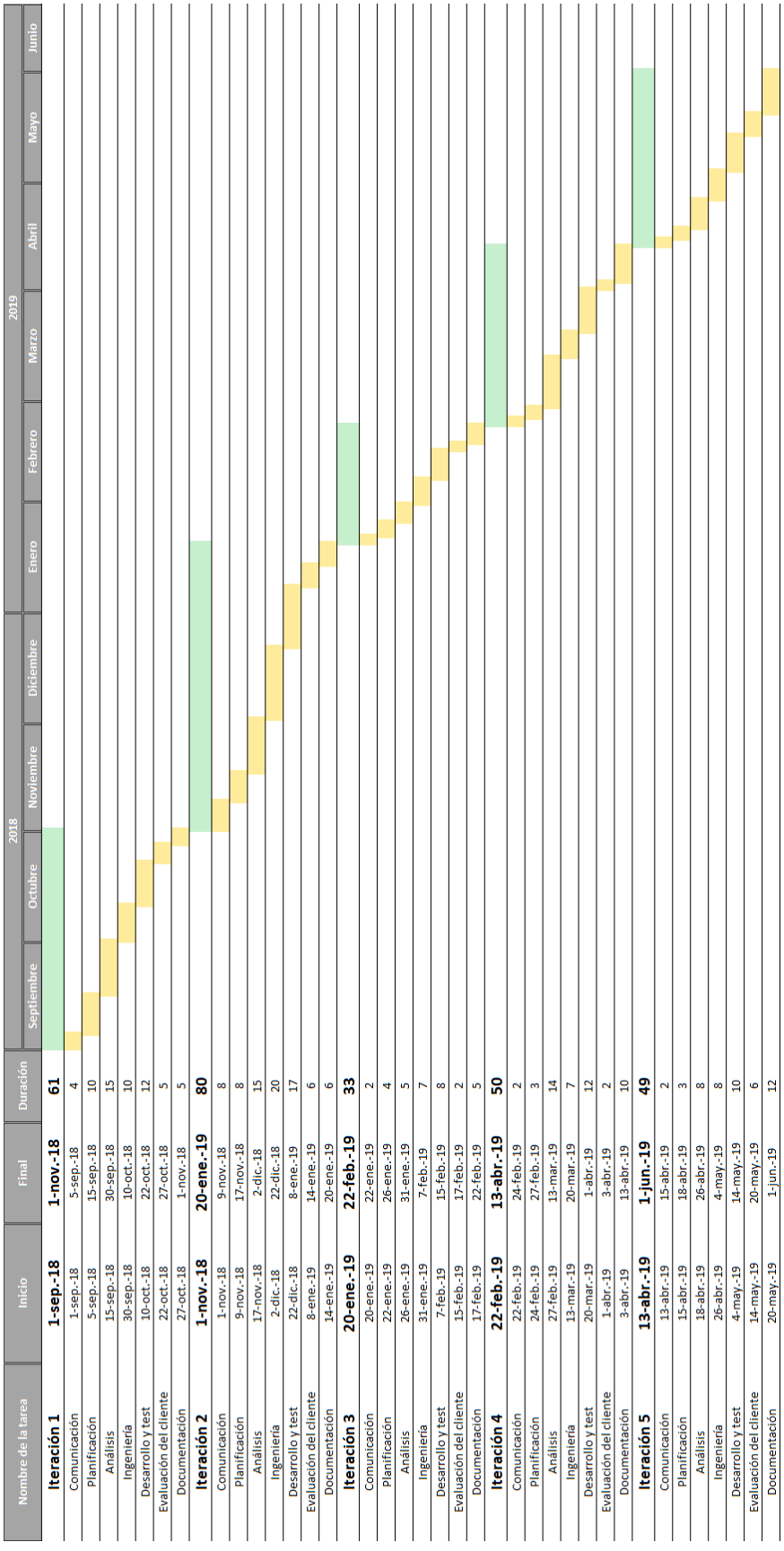


Fig. 7.1. Diagrama de Gantt del proyecto

7.2 Presupuesto

En esta sección se detallará el presupuesto del presente proyecto. En primer lugar, se mostrarán los costes del proyecto y, a continuación, se presentará la oferta presentada al cliente.

7.2.1. Costes del proyecto

La siguiente tabla muestra, de forma general, el presupuesto total que ha sido necesario para el proyecto:

TABLA 7.1. RESUMEN DEL PRESUPUESTO DEL PROYECTO

Autor	Elías Del Pozo Puñal
Tutor	Félix García Carballeira
Duración	9 meses.
Presupuesto	84.456,57€

Costes directos

En esta parte se van a mostrar los costes directos del proyecto. En primer lugar, se van a desglosar los costes del personal que ha formado parte en este TFG. En este proyecto existen cuatro tipos de roles: Director de proyecto, analista, desarrollador y probador. El tutor de este TFG es el director del proyecto, mientras que el estudiante que ha realizado el proyecto ha asumido el resto de roles mencionados anteriormente. Los costes resultantes han sido:

TABLA 7.2. COSTES EN RR.HH.

Rol	**Coste por hora**	**Horas**	**Total**
Director de proyecto	60€	78	4.680€
Analista	40€	171	6.840€
Desarrollador	30€	295	8.850€
Probador	20€	118	4.720€
Total			25.090€

Después de haber desglosado los costes directos relacionados con los recursos humanos, se analizarán los costes directos asociados a los equipos que han sido usados durante el desarrollo de este proyecto. Para obtener estos costes, se ha tenido en cuenta que los equipos informáticos tienen una vida media de 4 años, por lo que se devalúan un 25 % anual. De esta forma, se han obtenido los siguientes costes:

TABLA 7.3. COSTES EN EQUIPOS INFORMÁTICOS

Concepto	**Coste inicial (€)**	**Dedicación (meses)**	**Devaluación (%)**	**Coste total (€)**
Ordenador portátil	1250€	9	19 %	237,50€
ARCOS Tucán	89.501,60€	9	19 %	17.005,30€
Impresora	100€	5	8 %	8,00€
Total				17.250,80€

Los equipos que han sido usados para este proyecto se detallan a continuación:

- **Ordenador portátil:** MSI GE60 2PE Apache Pro, i7-4720HQ, 8GB RAM, 1TB.
- **ARCOS Tucán:** cluster de 10 PCs conectados en red, usado por el equipo de investigación Grupo de Arquitectura de Computadores de la Universidad Carlos III de Madrid.
- **Impresora:** Samsung Printer Xpress M2020W.

A continuación, se mostrará el material fungible que ha sido usado durante el desarrollo del proyecto, es decir, tinta de tóner, bolígrafos, folios, correctores y subrayadores:

TABLA 7.4. COSTES EN MATERIAL FUNGIBLE

Concepto	**Unidades**	**Precio Unidad (€)**	**Coste total (€)**
Tóner	2	25	50,00€
Paquete Folios	1	3	3,00€
Material de oficina	1	50	50,00€
Total			103,00€

TABLA 7.5. COSTES DIRECTOS

Concepto	**Coste (€)**
Costes en RR.HH.	25.090,00€
Costes en equipos informáticos	17.250,80€
Costes en material fungible	103,00€
Total costes directos	**42.443,80€**

Resumen de costes totales del proyecto

En esta parte se mostrará un resumen de los costes totales que ha supuesto la realización del proyecto. No obstante, cabe añadir que los costes indirectos del proyecto se ha estimado que son el 30 % de los costes directos obtenidos. Estos costes incluyen los gastos de luz, agua, mantenimiento y el acceso a internet.

TABLA 7.6. RESUMEN DE COSTES DEL PROYECTO

Concepto	**Coste (€)**
Total costes directos	42.443,80€
Total costes indirectos	12.733,14€
Total	**55.176,94€**

7.2.2. Oferta del proyecto

En esta sección, se incluye la oferta de este proyecto en la siguiente tabla. Cabe destacar que, a los costes obtenidos anteriormente, se le añade un margen del 10 % para posibles imprevistos, un beneficio del 15 % y, por último, se le añade un IVA del 21 % [49] según se establece en la legislación española. La oferta resultante es la siguiente:

TABLA 7.7. OFERTA DEL PROYECTO

Concepto	Incremento (%)	Valor parcial (€)	Coste agregado (€)
Costes del proyecto	-	55.176,94	55.176,94
Riesgo	10	5.517,69	60.694,63
Beneficio	15	9.104,19	69.798,82
IVA	21	14.657,75	84.456,57
Presupuesto total			**84.456,57**

7.3 Entorno socio-económico

Como se ha explicado a lo largo de la presente memoria, en la actualidad, es necesario manejar una gran cantidad de datos diariamente y ofrecerlos a los usuarios en el menor tiempo posible sin ningún fallo de integridad de la información. Por estas razones surgió la necesidad de crear la tecnología del Big Data [3] capaz de manejar este tipo de datos haciendo uso de un gran número de computadores conectados formando un cluster. Por eso, se fueron creando Sistemas de Ficheros que mejoraban los tiempos de procesado de lectura y escritura de grandes cantidades de datos, como puede ser Google File System o el Sistema de Ficheros de Hadoop llamado HDFS [1].

No obstante, también existen otras alternativas para el manejo de esta información como son las Base de Datos, capaces de almacenar un gran número de registros en muy poco tiempo. Como se ha querido demostrar en este proyecto, se pueden mezclar ambas tecnologías para proporcionar a las empresas la posibilidad de crear un Sistema de Ficheros haciendo uso de una Base de Datos, en este caso, de Apache Cassandra.

Además, si las empresas desean no utilizar uno, sino más de un Sistema de Ficheros[6], la existencia de la biblioteca MPI, y más concretamente de MPI-IO, promueve la posibilidad de interactuar con los datos almacenados en entornos diferentes gracias a lo que ofrece esta biblioteca. Es por esto que se decidió integrar parcialmente la interfaz que se ha desarrollado en MPI para probar que es posible utilizar esta tecnología para otros sistemas, como son las Bases de Datos, ya sean relacionales como no relacionales.

De este modo, se ofrece a las empresas una alternativa al uso de Sistemas de Ficheros para la manipulación de directorios y ficheros de muy diversos tamaños junto con

la posibilidad de interactuar entre más de un sistema que maneje grandes cantidades de información

Capítulo 8

Conclusiones y trabajo futuro

En este último capítulo de la memoria se van a describir unas conclusiones finales sobre el proyecto que se ha desarrollado y explicado en estas páginas (ver sección 8.1), además de unos posibles trabajos futuros que pueden completarse tras la presentación de este TFG (ver sección 8.2).

8.1 Conclusiones

En esta sección se va a analizar el objetivo principal que se querían conseguir y que se describió en la sección 1.2 de la Introducción, para determinar si se ha logrado ese objetivo o no.

El objetivo principal de este proyecto consistía en estudiar la aplicación de una Base de Datos para su uso como un Sistema de Ficheros Distribuido y Paralelo. Este objetivo genérico se tuvo que lograr mediante la realización de unos pasos intermedios que también se explicaron en la sección de los Objetivos mencionada anteriormente. Para poder determinar con seguridad que este objetivo se ha cumplido, se estudiarán en esta sección si esos pasos intermedios se han conseguido.

El primer paso a realizar para poder estudiar el uso de la Base de Datos en este ámbito consistió en diseñar e implementar una biblioteca de entrada y salida de mensajes que permitiese a los usuarios enviar y recibir información a la Base de Datos con la que se ha trabajado durante todo el proyecto. Para poder realizar estas tareas se tuvo que buscar información de varios ámbitos como puede ser el campo de las Bases de Datos, el campo

de los Sistemas Distribuidos y paralelos, el campo del Big Data y, además, el campo de los Sistemas Operativos. Este diseño va en relación con los siguientes pasos que se van a detallar a continuación sobre la implementación de las interfaces para POSIX y MPI-IO.

Para poder determinar si de verdad un usuario es capaz de manejar ficheros y directorios de la misma manera que lo hace UNIX con el estándar POSIX, se tuvieron que elegir una serie de funciones del estándar y adaptarlo a la biblioteca antes diseñada para que permitiese a los usuario poder realizar varias operaciones de entre todas las ofrecidas por POSIX.Cabe decir que para la realización de esta interfaz no se implementaron todas las funciones que define el estándar POSIX, puesto que solo se quería determinar su viabilidad para el tratamiento de este tipo de elementos, por eso se tuvieron en cuenta un conjunto reducido de funciones.

De igual forma que para POSIX, la ruta seguida para diseñar una interfaz para MPI-IO con la biblioteca desarrollada es muy similar. Para ello, ha sido necesario buscar bastante más información que con respecto a POSIX, puesto que no existe tanta documentación, como sí ocurre con Apache Cassandra o los Sistemas de Ficheros. No obstante, ha sido posible integrar varias de las funciones que se han desarrollado en este TFG para poder determinar si era posible integrarlo en esta interfaz y poder usarlo con otro tipo de Sistemas de Ficheros que soportase MPI-IO, como puede ser NFS.

Por último, para determinar si se ha conseguido cumplir el objetivo principal de este trabajo, se tuvo que realizar una serie de pruebas de rendimiento a las diferentes interfaces implementadas, junto con un Sistema de Ficheros al uso, como es HDFS. Para poder conseguirlo, como se ha comprobado en el capítulo de Verificación, validación y evaluación (sección 6.2), ha sido necesario crear un conjunto de pruebas amplio, con diversos tamaños de bloque y diversos tamaños de fichero, que permitiesen determinar la flexibilidad y la capacidad que tienen ambos sistemas ante el tratamiento los datos almacenados en ellos. Es necesario indicar que, aunque los tamaños de los ficheros utilizados en este trabajo sean más pequeños que los que se puedan trabajar en grandes empresas como Google, entre otros, sí que ayuda a tener una idea del comportamiento que puede tener si se llegara a utilizar en entornos de mayor tamaño. Cabe añadir que, como se ven en las pruebas, Apache Cassandra puede servir para algunos casos como Sistema de Ficheros, pero, por otro lado, no está a la altura de aquellos sistemas que están verdaderamente

optimizados para manejar estos tipos de datos.

De esta forma, se puede concluir que el objetivo principal para determinar si se puede utilizar una Base de Datos para el desarrollo de un Sistema de Ficheros Distribuido y Paralelo se ha conseguido satisfactoriamente.

8.2 Trabajo futuro

Después de haber finalizado el presente TFG, se procede a mostrar una serie de trabajos o líneas futuras, que pueden servir para mejorar este proyecto:

- Integración total de la interfaz desarrollada en MPI-IO.
- Evaluación del rendimiento de la interfaz desarrollada en un entorno cloud, como puede ser el entorno que ofrece la empresa Amazon, llamado Amazon EC2.
- Integración de la interfaz desarrollada en Mimir, que es una biblioteca que permite a los usuarios la programación de funciones Map y Reduce dentro de entornos MPI.
- Realizar un módulo que permita incluir la biblioteca desarrollada dentro del núcleo de Linux utilizando FUSE (Filesystem in User Space).

Glosario

Apache Cassandra Base de Datos no relacional de código abierto desarrollado por Apache Software Foundation en 2008 y que se caracteriza por usar el modelo de almacenamiento (clave,valor). 2, 14, 18, 19, 25, 27, 29, 32, 46, 52, 62, 83, 108–110, 112, 122, 123

Apache Software Foundation Organización fundada en junio de 1999 en Estados Unidos sin fines de lucro cuyo objetivo principal consiste en dar soporte a los proyectos de software libre. 2, 11, 46

Base de Datos Colección organizada de datos, normalmente almacenadas electrónicamente en una computadora. 1–3, 7, 8, 14–16, 18–20, 25–28, 32–35, 37–40, 42–44, 46, 49, 51–54, 56, 57, 61, 62, 64–76, 78, 79, 81, 85–97, 109, 110, 114, 115, 123

Big Data Conjuntos de datos o combinaciones de ellos que necesitan un tratamiento especial para poder manejar su tamaño y complejidad usando técnicas distribuidas. 1–3, 7, 10

C Lenguaje de programación desarrollado en 1972. 21, 28, 29

C++ Lenguaje de programación desarrollado en 1979 como una extensión del lenguaje C. 28

cluster Conjuntos de ordenadores unidos normalmente por una red y que se comportan como una única computadora. 3, 14–16, 18, 20, 27, 32, 61, 109, 110, 114, 126

Datanode Elemento o componente de Hadoop cuyo objetivo es el de almacenar los bloques de datos de los ficheros que se almacenan en el Sistema de Ficheros HDFS. 11

driver Conector que permite la comunicación con una aplicación a través de un lenguaje de programación específico. 27–29, 46, 62

Google Multinacional americana tecnológica especializada en productos y servicios de Internet fundada en 1998. 2, 10, 11

IVA Impuesto sobre el Valor Añadido de España desde 1992. 127, 128

Java Lenguaje de programación orientado a objetos y de propósito general desarrollado en 1996 por Sun Microsystems. 28, 62

Map Función perteneciente al modelo de programación Map-Reduce cuyo objetivo es procesar pares (clave,valor) para generar un conjunto de datos intermedio de la forma (clave,valor). 12, 13

Map-Reduce Modelo de programación que presta soporte a la computación paralela sobre grandes conjuntos de datos que se encuentran en grupos de computadoras. 12, 14

Nodejs Entorno de ejecución de código abierto cuya primera aparición fue en 2009. 28

NoSQL Sistema de gestión de Base de Datos que se caracteriza por no hacer uso del lenguaje de consultas SQL y por no tener la necesidad de almacenar la información en tablas como sí hacen los sistemas gestores de Bases de Datos relacionales. 2, 18

Python Lenguaje de programación interpretado a alto nivel que apareció en 1990. 28

Reduce Función perteneciente al modelo de programación Map-Reduce cuyo objetivo es combinar los valores obtenidos de la función Map con la misma clave y devolver el resultado final de la operación. 12–14

Sistema de Ficheros Agrupación de estructuras de datos que controla cómo la información se almacena y se obtiene. 2, 3, 8–12, 21, 25, 43, 52, 61, 109

Software libre Libertad de los usuarios para modificar, ejecutar, copiar, distribuir y mejorar el software de modo más preciso y sin ningún coste. 2, 46

UNIX Sistema Operativo multitarea y multiusuario desarrollado en 1969. 11, 21, 25, 27

Siglas

ARCOS Grupo de Arquitectura de Computadores, Universidad Carlos III de Madrid. 126

BBDD Bases de Datos. 19, 69

CPU Central Processing Unit, Unidad Central de Procesamiento. 30

CQL Cassandra Query Language, Lenguaje de consultas de Cassandra. 19, 20

GFS Global File System, Sistema de Ficheros Global. 10

HDFS Hadoop Distributed File System, Sistema de Ficheros distribuido, escalable y portátil usado para el framework Hadoop. 11, 12, 25, 83, 109, 110, 114–118, 121, 123, 132, 160

IEEE Institute of Electrical and Electronics Engineers, Insituto de Ingeniería Eléctrica y Electrónica. 3, 21, 29

IP Internet Protocol, Protocolo de Internet. 27, 32, 85–97, 141–143

MPI Message Passing Interface, Interfaz de Paso de Mensajes. 3, 7, 22, 45, 98, 101–103

MPI-IO Message Passing Interface - Input/Output, Interfaz de Paso de Mensajes de Entrada/Salida aplicada al acceso a los Sistemas de Ficheros. 3, 22, 23, 132, 159, 160

NFS Network File System, Sistema de Ficheros de red. 9, 132, 160

PHP Hypertext Preprocessor. Lenguaje de programación cuya función principal consiste en el desarrollo web de contenido dinámico para el servidor. Surgió en 1995. 28

POSIX Portable Operating System Interface, Interfaz de Sistema Operativo Portable. Norma escrita por el IEEE que define una interfaz estándar del Sistema Operativo. 3, 7, 21, 25, 28, 29, 45, 49, 51–53, 55, 97, 121, 132, 159

RPC Remote Procedure Call, Llamada a Procedimiento Remoto. 9

SQL Structured Query Language, Lenguaje de Consulta Estructurado. 18–20

TFG Trabajo de Fin de Grado. 121–123, 125

Anexo A

Instalación de Cassandra

En este anexo se va a mostrar una explicación sobre cómo se ha procedido para instalar la Base de Datos Apache Cassandra en un cluster. Se supondrá que se instalará la Base de Datos en cuatro nodos, pero en el caso de que se instalase en más nodo el procedimiento es similar. Esta explicación supone que se instala la versión 3.11.4.

A Configuración

A.1. Prerrequisitos

Antes de proceder a instalar la Base de Datos es necesario que los nodos cumplan los siguientes requisitos:

1. Comprobar que los nodos que se van a utilizar están conectados entre sí permitiendo su comunicación mediante el comando ssh.
2. Tener instalado en cada nodo la versión 8 de Java. Además, es necesario que esté establecida la variable de entorno JAVA_HOME que indica dónde está instalado Java. En este caso, la ruta es /usr/lib/jvm/java-8-openjdk-amd64/jre.
3. Para poder usar la interfaz de comandos de Apache Cassandra es necesario tener instalado Python 2.7.
4. Tener instalado Ubuntu 16.04 o superior.
5. Tener instalado Libuv, que es una biblioteca en C multiplataforma que proporciona soporte para operaciones asíncronas E/S.

A.2. Descarga de binarios de Apache Cassandra

A continuación, se procede a explicar el proceso de descarga de los binarios de Cassandra para poder realizar la posterior instalación:

1. Usar el comando cd para dirigirnos al directorio donde queramos instalar Cassandra.

2. Usar el comando wget para obtener los binarios de la siguiente forma:

wget http://apache.rediris.es/cassandra/3.11.4/apache-cassandra-3.11.4-bin.tar.gz

3. Usar el comando *tar -xzf apache-cassandra-3.11.4-bin.tar.gz* para descomprimir el archivo descargado en el directorio deseado.

4. Modificar el nombre del archivo descargado al nombre deseado. En este caso el nombre que se usará será "cassandra" por comodidad.

5. Por último, establecer la variable de entorno PATH introduciendo la carpeta de instalación de Cassandra: /directorio_descarga/cassandra/bin

Con esta variable de entorno ya es posible utilizar Cassandra en un único nodo. Para poder hacerlo en un cluster, se explicará en la siguiente sección.

Por otro lado, para poder utilizar el conector o driver necesario a la hora de hacer la interfaz de C, será necesario realizar los siguientes pasos:

6. Descargar los siguientes paquetes .deb con el comando wget:

wget https://downloads.datastax.com/cpp-driver/ubuntu/16.04/cassandra/v2.12.0/cassandra-cpp-driver-dbg_2.12.0-1_amd64.deb
wget https://downloads.datastax.com/cpp-driver/ubuntu/16.04/cassandra/v2.12.0/cassandra-cpp-driver-dev_2.12.0-1_amd64.deb
wget https://downloads.datastax.com/cpp-driver/ubuntu/16.04/cassandra/v2.12.0/cassandra-cpp-driver_2.12.0-1_amd64.deb

- Una vez descargados estos paquetes, conviene instalarlos con el comando dpkg de la siguiente forma:

dpkg -i cassandra-cpp-driver-dbg_2.12.0-1_amd64.deb
dpkg -i cassandra-cpp-driver-dev_2.12.0-1_amd64.deb
dpkg -i cassandra-cpp-driver_2.12.0-1_amd64.deb

A.3. Configuración de la instalación de Apache Cassandra

En esta sección se va a explicar la configuración que deben tener cada una de las máquinas, indicando los parámetros que se deben introducir en el fichero cassandra.yaml que se encuentra en el directorio: directorio_descarga/cassandra/conf.

Lo primero que hay que saber son las direcciones IP que tiene cada nodo del cluster, puesto que son necesarias para que sepa la Base de Datos entre qué nodos se deben conectar. En este caso, los nodos que han sido usados van del compute-1-1 al compute-1-4, cuyas direcciones IP son 10.0.40.9, 10.0.40.10, 10.1.0.9 y 10.1.0.10.

El primer paso para configurar los nodos consistiría en indicar el nombre del cluster, de tal forma que todos los nodos que lo constituyan se llamen igual, si no, se formarían diferentes cluster con diferentes nombres. Para poner el nombre se debe escribir en el campo cluster_name:

- cluster_name:'nombre del cluster'

El siguiente paso a modificar tiene que ver con el tamaño del segmento de commitlogs o registros de confirmación, cuya función es almacenar los datos que recibe de las consultas hasta el tamaño que se le indique, en MB, antes de poder almacenarlos en las tablas. Es por eso que es importante modificar este campo, indicando que pueda almacenar tamaños grandes de consultas, puesto que se pueden manejar ficheros con un tamaño de bloque que puede ser bastante grande, por lo tanto, lo recomendable es poner lo siguiente:

- commitlog_segment_size_in_mb: 1024

Lo siguiente sería indicar las direcciones IP que conformarían el cluster. Para ello, en el campo seeds se deben incluir las direcciones IP, separadas por comas para que así la

Base de Datos pueda comunicarse entre los nodos y conformar el cluster en su totalidad. Si lo que se desea es ampliar en algún momento el cluster, se deben incluir aquí cuantas IPs se deseen, e indicarse en cada uno de los nodos.

- seeds: "10.0.40.9,10.0.40.10,10.1.0.9,10.1.0.10"

Además, para cada uno de los nodos, de forma individual, se deben indicar sus propias direcciones IP para los campos de listen_address (campo que indica la dirección IP a la que se tienen que conectar el resto de los nodos para la construcción del cluster) y rpc_adress (campo que indica la dirección IP a la que puede conectarse el cliente para acceder a la Base de Datos).

- listen_address: 10.0.40.9 (cada nodo su dirección IP)
- rpc_address: 10.0.40.9 (cada nodo su dirección IP)

Para evitar que la Base de Datos rechace la consulta por tardar mucho en enviarse el bloque de datos de un fichero por su longitud, es recomendable modificar los tiempos de espera del cluster y poner unos tiempos lo suficientemente altos como para que no lo rechace. Es por ello que se recomienda modificar los siguientes campos:

- read_request_timeout_in_ms: 400000
- range_request_timeout_in_ms: 100000
- write_request_timeout_in_ms: 400000
- counter_write_request_timeout_in_ms: 50000
- cas_contention_timeout_in_ms: 10000
- truncate_request_timeout_in_ms: 600000
- request_timeout_in_ms: 300000

Por último, es recomendable que, para la comunicación entre nodos se utilice el "gossip'' y se puedan enviar entre ellos réplicas de los datos que se almacenen, se use como endpoint_snitch (campo que indica la forma de enviarse réplicas entre nodos):

- endpoint_snitch: GossipingPropertySnitch

B Ejecución

Con los pasos realizados ya se puede utilizar la Base de Datos, pero primero habría que iniciar los nodos individualmente de la siguiente forma:

cassandra -f >directorio_deseado/nombre_fichero &

De esta forma, las trazas que se obtengan de la ejecución en segundo plano de Cassandra en cada nodo, se almacenarán en el fichero auxiliar que se designe.

Finalmente, para poder conectarse a la shell de la Base de Datos y, por ende, hacer consultas, se debe utilizar el comando cqlsh indicando la dirección IP del nodo al que se quiera conectar, tal y como se describe a continuación:

cqlsh dirección_IP

Anexo B

Instalación de Hadoop

En este anexo se va a mostrar el proceso de instalación y ejecución de Hadoop. La configuración de Hadoop no se ha llevado a cabo porque se ha reutilizado una configuración ya existente de previos trabajos y ha sido trasladado al entorno donde se encuentra la Base de Datos.

A Configuración

A.1. Prerrequisitos

Antes de empezar a obtener los binarios y de tener clara la arquitectura que seguirá HDF [1] es necesario verificar que se cumplen los siguientes requisitos:

1. Comprobar que los nodos que se van a utilizar están conectados entre sí permitiendo su comunicación mediante el comando ssh.
2. Tener instalado en cada nodo la versión 8 de Java. Además, es necesario que esté establecida la variable de entorno JAVA_HOME que indica dónde está instalado Java. En este caso, la ruta es /usr/lib/jvm/java-8-openjdk-amd64/jre.

A.2. Arquitectura del cluster de Hadoop

A continuación, se explicará la arquitectura que sigue Hadoop para su uso:

- Namenode: Componente cuya función principal es gestionar el Sistema de Ficheros

y de saber en todo momento dónde se encuentra almacenado cada bloque de datos de los ficheros.

- Datanode: Componente que se encarga de almacenar los bloques de datos.
- ResourceManager: Elemento que administra las aplicaciones yarn que se están ejecutando.
- NodeManager: Componente que gestiona la ejecución de tareas que se realizan en el nodo.

A.3. Descargar binarios de Hadoop

Para poder usar Hadoop es necesario descargarse los binarios realizando los siguientes pasos:

1. Usar el comando *cd* para dirigirnos al directorio donde se quiere instalar Hadoop.
2. Usar el comando *wget* para descargarse los binarios en el directorio actual de la siguiente forma:

   wget http://apache.rediris.es/hadoop/common/hadoop-2.9.1/hadoop-2.9.1.tar.gz

3. Usar el comando *tar* para descomprimir el archivo descargado:

   tar -xzf hadoop-2.9.1.tar.gz

4. Modificar el nombre del archivo descargado al nombre deseado. En este caso el nombre que se usará será “hadoop” por comodidad.
5. Por último, se deben establecer diferentes variables de entorno, siendo la primera de ellas la variable PATH, introduciendo las carpetas de instalación: hadoop/bin y hadoop/sbin. Además, también se deben poner las siguientes variables de entorno para una correcta ejecución de Hadoop:
   - HADOOP_INSTALL: Ruta a la carpeta de instalación donde se ha descargado Hadoop anteriormente.

- HADOOP_MAPRED_HOME: $HADOOP_INSTALL
- HADOOP_HDFS_HOME: $HADOOP_INSTALL
- HADOOP_YARN_HOME: $HADOOP_INSTALL
- HADOOP_COMMON_LIB_NATIVE_DIR: $HADOOP_INSTALL/lib/native
- HADOOP_OPTS: "-Djava.library.path=$HADOOP_INSTALL/lib"

De esta forma ya se podría utilizar Hadoop sin ningún problema. No obstante, si se quieren hacer pruebas de Hadoop utilizando el lenguaje de programación C, como se ha hecho en el presente trabajo para las pruebas de rendimiento del sistema, hay que hacer uso de la biblioteca "libhdfs". Para ello, se debe añadir una nueva variable de entorno llamada LD_LIBRARY_PATH al fichero .bashrc de la siguiente forma:

- LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/$HADOOP_INSTALL/lib/native

Como esta biblioteca llama internamente a Java, hay que modificar el classpath introduciendo todos los ficheros .jar correspondientes para su correcto funcionamiento. Para ello se deben seguir los siguientes pasos:

1. Insertar en el .bashrc las rutas a los .jar al classpath con el siguiente comando:

   hadoop classpath –glob ».bashrc

2. Abrir el fichero .bashrc y añadir la variable de entorno CLASS_PATH antes de las rutas a los .jar introducidos: CLASS_PATH=$CLASS_PATH:Rutas_a_los_jar.

B Ejecución

Una vez que se ha instalado correctamente y configurado de igual forma que con los anteriores proyectos que han utilizado este Sistema de Ficheros ya se puede ejecutar sin problemas. No obstante, lo primero que se debe hacer es formatear el Namenode:

hdfs NameNode -format

Tras esto, se debe iniciar Hadoop en cada uno de los nodos y comprobar después, con el comando *jps* si se están ejecutando correctamente. Después, se debe crear una carpeta dentro del Sistema de Ficheros HDFS [1] con el siguiente comando:

hdfs dfs -mkdir -p /user/nombre_directorio

Creado el directorio deseado, ya se pueden realizar diferentes operaciones de creación, modificación y borrado tanto de ficheros como de directorios en el Sistema de Ficheros.

Appendix C
Extended Abstract

In this appendix, an extensive explanation of the project that has been carried out will be provided, beginning with a brief introduction. It will then describe the reasons for the project and the objectives to be achieved. In addition, it will explain the decisions that have been taken for the development of the File System, together with the evidences that have been obtained comparing them with the performance of another File System such as HDFS [1]. Finally, some brief conclusions of the project will be given.

A Introduction

Today, the amount of information that users handle throughout the day has increased considerably, thanks to the continuous advancement of technology and storage systems available on the network. Information of any kind is generated, from the purchase of a product over the Internet, to data created when users comment on social networks or communicate with other users through instant messaging services.

In order to offer users the ability to send and receive information in a reduced time, it is necessary to use different data processing techniques along with various File Systems or Databases optimized to handle the information correctly. In order to be able to process the data, a set of techniques called Big Data [3] emerged that allow the processing of large sets of information in a distributed and efficient manner. As these techniques were exclusively distributed, the need arose to create Distributed File Systems.

For this project, it started from the idea of being able to manage large amounts of data in a distributed way in order to develop a File System using a Database called Apache

Cassandra. This Database is oriented to the storage of large datasets offering users low latency in operations.

To finish, once the interface of the File System is developed on the Database, a library is used for Distributed Systems designed so that the applications that use it take full advantage of the power offered by the processors. This interface, known as MPI[17], allows the use of concurrent programming techniques between several processes thanks to semaphores or locks.

B Motivation

The need to store all the information that users handle daily and recover the data in the shortest time possible without losing its integrity is what has led to want to carry out this project as a prototype. In this way, it can be determined if there are other ways to implement a Distributed and Parallel File System in which different users can access the same data simultaneously.

In addition, with other techniques that currently exist, you can integrate the interface that you want to develop into another message pass-through interface known as MPI-IO[18], which offers the possibility of interacting with different File Systems.

C Objectives

In this project, different objectives are shown, among which the following is to be highlighted in a more general way: **study of the application of a NoSQL Database for use as a Distributed and Parallel File System**. In order to achieve this goal, the following steps must be carried out in this project:

- Design and implementation of a basic input and output library to make use of the Apache Cassandra Database.
- Implementation of an interface based on POSIX with the input and output library for the Database.
- Implementation of an interface based on MPI-IO with the input and output library for the Database.

- Obtaining the performance offered by the library developed along with the performance of a file system such as Hadoop Distributed File System (HDFS).

D Summary of the defined design

In this section, it will proceed to describe the design that has been chosen to meet the objectives of developing an interface based on POSIX on Apache Cassandra, starting with the initial structure that will present the Database (subsection D.1), followed by the components that make up this interface to develop that allows users to access the files they want (subsection D.2).

D.1. Summary of the Apache Cassandra Structure for the File System

This section will explain the decisions that have been made for the structure that will follow the Database so that users can create, modify and delete both files and directories. It is worth mentioning that in order to develop the interface a series of restrictions that the Database has when naming directories have been taken into account. More specifically, the paths of directories and files have had to be modified, changing the "/" character to "_" by design decision.

Taking this restriction into account, in the first instance there are two workspaces in the Database, "keyspaces", one to store the directories that are created with the elements that each one has saved (keyspace "dir"), having the root directory from the beginning, and another to store initially the files in the root directory (keyspace "_"). However, as you use it, not only will there be a single keyspace, since as you create directories, the number of "keyspaces" will increase.

On the one hand, the "keyspace" dir will store all the directories that are created during the execution of the program. These directories will be tables that will follow the name "dir._absolute_path_directory" where the second part of the name indicates the complete path of the created directory. This table, in turn, presents different fields, which will be the name of the directory without being modified (dirname), the absolute path of the parent directory ("..") and the stat field that will store the metadata of the directories

created within it. In addition, as files are created in the directory, this table will increase, creating new columns that will have the name of the file that has been created. The following figure shows the explained structure.

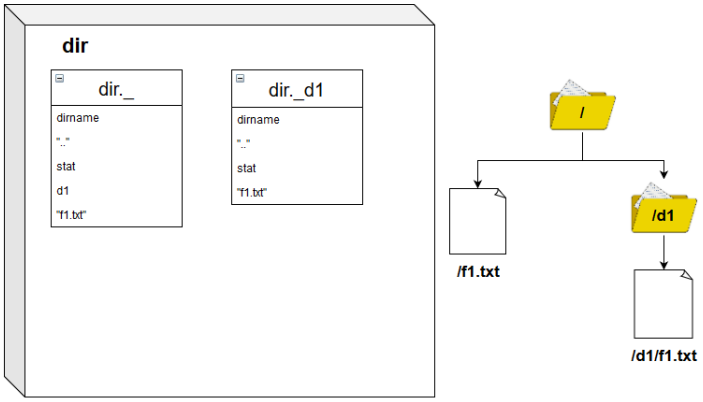


Fig. D.1. *Example keyspace dir*

On the other hand, the second "keyspace" that exists initially in the Database is the root directory "_", directory in which all the files that users want to save in it will be stored, as well as all the directories that they want to create. The operation of this "keyspace" will be the same as the subsequent ones created when you want to create a new directory in the Database.

When a user wants a new file in a certain directory, a table will be created where the blocks of data received will be stored and it will have the name "_absolute_path_directory. filename". Each file will be formed by this table and will have the following fields:

- Block: Field that will indicate the data block number. It will have an extra number that will be -1, row in which the metadata of the file will be stored.
- Opened: Field that will indicate the number of users that have opened the file.
- Stat: Field that stores all the relevant metadata of the file to which it refers and that will be used for the stat function explained previously in the Implementation part (section 5.2.2).
- Bytes: Number of bytes stored in the data block.
- Text: Content of the data block.

- N_blocks: Number of total blocks in the file.

The fields opened, stat and n_blocks will only be used to obtain related information from the associated file so only this information will be written in a single row that will be the one with block number -1, while the other rows will store, from block number 0 to n-1, the block of data, the number of bytes that have been saved in that row and the associated block number. An example of the structure that the File System to be designed will follow is shown in the following figure.

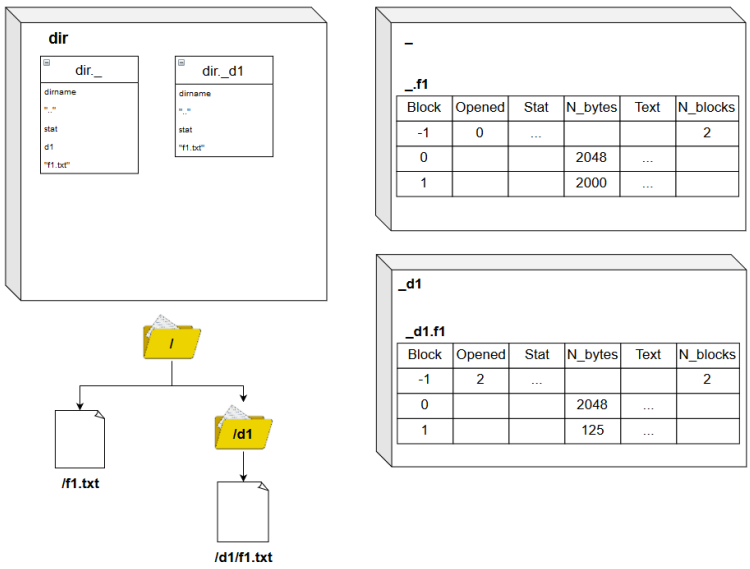


Fig. D.2. *Example of keyspace and file structure*

D.2. **Summary of the POSIX-based access interface to the Database**

After having described in depth the designed structure that will follow the Database, the process followed to design the user interface for the management of files and directories based on POSIX will be explained.

This library, which will be designed and implemented, will offer functionalities for access to files and directories similar to those of the POSIX standard. As it is a library of functions at user level, the names of the functions to be developed will have the prefix c_ , in such a way that it can be differentiated from normal POSIX calls.

This interface will be divided into three sets. The first set will bring together those functions that allow the manipulation of files, the second set will bring together the functions that will allow users to manipulate directories and, finally, the last set of functions

will be those that offer users the ability to know the metadata of the files and directories stored in the Database.

File manipulation functions

Next, the functions to be implemented that allow users to manipulate regular files will be explained.

- **C_creat**: This function allows you to create a regular file. In this function, the different permissions of writing, reading and execution that the file will have for the three types of users that could access the file must be introduced. These three groups are: the user, the group and others. These permissions will be introduced in octal format, in the same way as in POSIX.
- **C_open**: This function allows the opening of a database file for its later manipulation either to read or write in it. The method receives, as in POSIX, several arguments, which are the absolute path of the file, and the flags or opening modes of the file. With respect to this last field, the following modes have been supported: O_CREAT, O_TRUNC, O_APPEND and O_RDWR.
- **C_read**: Function that will read the previously opened file. In order to read it, the method receives several parameters that are the associated file descriptor, the buffer where the data will be stored and the number of bytes to be read from the file.
- **C_write**: Function that will write the previously opened file. In order to be able to write in it, the method receives several parameters that are the associated file descriptor, the buffer where the data you want to store in the file and the number of bytes you want to write in it.
- **C_close**: Function that allows the closing of the file if it has been opened previously by the process.
- **C_unlink**: Function that allows deleting the file from the Database if you have writing permissions on the associated file and, furthermore, there is no other user accessing the same file you want to delete.

- **C_lseek**: Function that allows on a file, in a local way, to modify the pointer of position for its later writing or reading of the information stored in it. In order to move the pointer, the method receives different arguments, which are the file descriptor obtained after opening, the number of bytes you want to move (offset) and, finally, where you want to start moving the pointer (whence). This last parameter has three different modes or types: SEEK_CUR, SEEK_SET and SEEK_END.

Directory manipulation functions

After explaining the functions associated with the files, the functions to be implemented that allow users to manipulate directories will be explained.

- **C_mkdir**: This function allows you to create a directory. In this function, you must enter the different permissions for writing, reading and execution that the directory will have for the three types of users that could access it. These three groups are: the user, the group and others. These permissions will be introduced in octal format, in the same way as in POSIX.
- **C_rmdir**: Function that will remove from the Database the directory indicated by the user by entering the user's absolute path. This directory will be deleted if there is no element inside it and, in addition, you have permissions in the directory to be modified.
- **C_opendir**: Function that will allow to open the directory that the user indicates in the argument of the function for its later reading of elements. It will be possible to read the desired directory if you have reading permissions.
- **C_readdir**: Function that will allow reading the directory previously opened by the user in order to obtain the elements stored in it.
- **C_closedir**: Function that will close the directory previously opened by the user.

Metadata retrieval functions

Finally, the functions related to obtaining metadata from files and directories will be defined.

- **C_stat**: Function that allows to obtain metadata from files and directories stored in the Database. To know this information, the user will have to enter the absolute path of the file or directory and a structure where the data associated with these elements will be stored.
- **C_fstat**: Function that allows to obtain the metadata of the files. In order to receive this information, the user must have the file open in order to be able to use this function because it has as an argument an open file descriptor together with a structure in which the associated information must be stored.

E Apache Cassandra Interface Implementation Summary

In this section of the appendix, a summary of how the Database was initially built and how the interface will be implemented will be explained. The pseudocode of the interface, to avoid repeating the same lines of pseudocode, it is recommended to read section 5.2.2 of this report.

E.1. Initial construction of Apache Cassandra

In this part, the procedure followed to structure the Database will be shown in such a way that, when implementing the access interface, file and directory manipulation operations can be done correctly. To do this, the first step consists of creating two "keyspace", one to save all the directories created at runtime (dir) and another to have, initially, the root directory ("_"). To do this, the steps that have been followed have been as follows:

1. Creation of keyspace dir: CREATE KEYSPACE IF NOT EXISTS dir WITH replication='class': 'SimpleStrategy', 'replication_factor': '1' ;
2. Creation of keyspace "_": CREATE KEYSPACE IF NOT EXISTS"_" WITH replication='class': 'SimpleStrategy', 'replication_factor': '1' ;
3. Creation of the root directory table:: CREATE TABLE IF NOT EXISTS dir."_" (dirname text,".."text, uid text, mode text, stat text, PRIMARY KEY(dirname));
4. Insertion of the initial metadata of the root directory: INSERT INTO dir."_" (dirname,"..",uid,mode,stat) VALUES ('/','/','0','0755','stat');

As you can see, it has been used as an example that the replication factor of the files that are saved is 1, that is to say, that only one copy is kept in the Database. If you want to have more copies, it will be necessary to modify the field “replication_factor” to the desired value.

E.2. **Implementation of the interface**

In order to develop the interface and be able to offer the user the ability to interact with the Database to do file and directory management operations, it has been necessary to choose a programming language close to the POSIX standard to which it has to resemble and is the C language.

In addition, in order to achieve this connection between both ends of the application, it will be necessary to use a connector provided by the company Datastax that offers a library of functions that allows such communication to make queries as desired. This connector will allow access to the Database using only Cassandra’s own query language, CQL3 [31]. The interaction between the components that are part of a normal execution between the user and the Database is the following:

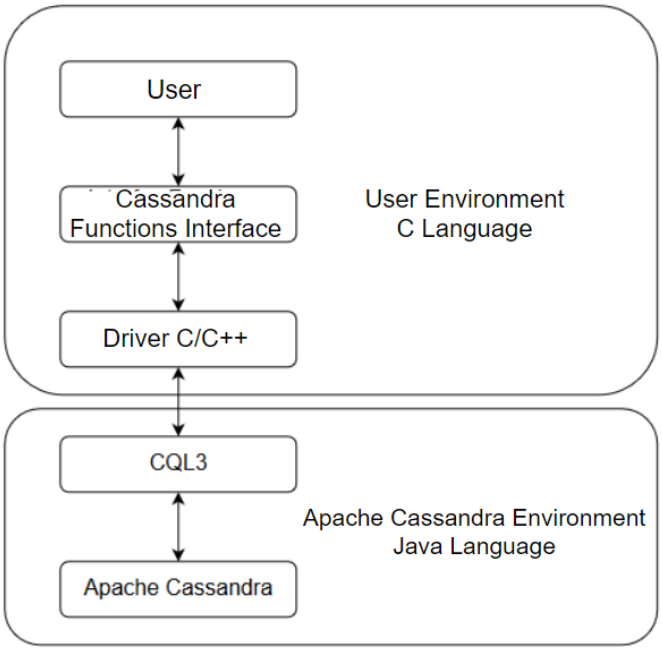


Fig. E.3. *Component interaction*

F Summary of the test perfomed

In this section, an explanation of the results obtained and shown in section 6.2 is going to be made in a very summarized form. To avoid repeating the same data, general conclusions will be given on the transfer rates offered by both HDFS and Apache Cassandra, as well as the rates offered by Cassandra if the interface were integrated into MPI.

First, if you compare the use of Cassandra between 4 and 8 nodes for both writing and reading, the best transfer rates offered are those that use a block size of 1MiB along with a cluster of 4 nodes. However, the performance offered by the cluster of 8 nodes both with replication factor 1 and 3 is not far from the other speed rates.

Secondly, between HDFS and Cassandra there is a noticeable difference in performance, because HDFS far exceeds the writing and reading of files with a single process used of 512MiB and 1024MiB, although with small files, such as the 1MiB used, Cassandra beats HDFS. On the other hand, if several processes are used for reading and writing, although HDFS does not allow parallel writing, it can be seen in the figures 6.15 and 6.16 that Apache Cassandra, with a cluster of 8 nodes, manages to equalize and even improve the performance offered by HDFS.

Finally, if the interface designed in MPI were to be integrated, the transfer rates that Cassandra would offer are not much different from the non-integrated interface, as can be seen in the figures 6.17 and 6.18. It can therefore be concluded that it is feasible to integrate the interface developed in a multiple File System environment, as provided by the MPI-IO interface.

G Conclusions and future work

This last chapter of the document will describe some final conclusions about the project that has been developed and explained in these pages (see section G.1), as well as some possible future works that can be completed after the presentation of this project (see section G.2).

G.1. Conclusions

In this section the main objective to be achieved, described in section C, will be analysed to determine whether that objective has been achieved or not.

The main objective of this project was to study the application of a Database for use as a Distributed and Parallel File System. This generic objective had to be achieved by carrying out intermediate steps which were also explained in the Objectives section mentioned above. In order to be able to determine with certainty that this objective has been met, it will be studied in this section whether these intermediate steps have been achieved.

The first step to be taken in order to study the use of the Database in this field consisted of designing and implementing a library of incoming and outgoing messages that would allow users to send and receive information to the Database with which they have worked throughout the project. In order to be able to carry out these tasks, information had to be sought from various fields such as the Database field, the Distributed and parallel Systems field, the Big Data field and, in addition, the Operating Systems field. This design is related to the following steps that will be detailed below on the implementation of interfaces for POSIX and MPI-IO.

In order to determine if a user is really capable of handling files and directories in the same way as UNIX does with the standard POSIX, a series of standard functions had to be chosen and adapted to the library previously designed to allow the user to perform several operations among all those offered by POSIX.It must be said that for the realization of this interface not all the functions defined by the standard POSIX were implemented, since it was only wanted to determine its feasibility for the treatment of this type of elements, so a reduced set of functions was taken into account.

Just like for POSIX, the path taken to design an interface for MPI-IO with the developed library is very similar. For this, it has been necessary to look for quite a bit more information than with respect to POSIX, since there is not as much documentation as there is with Apache Cassandra or File Systems. However, it has been possible to integrate several of the functions that have been developed in this project in order to determine if it was possible to integrate it in this interface and to use it with other types of File Systems

that support MPI-IO, such as NFS.

Finally, in order to determine whether the main objective of this work has been achieved, a series of performance tests had to be carried out on the different interfaces implemented, together with a custom File System, such as HDFS. In order to achieve this, as has been proven in the Verification, Validation and Evaluation chapter (section 6.2), it has been necessary to create a comprehensive set of tests, with different block sizes and different file sizes, in order to determine the flexibility and capacity of both systems with regard to the processing of the data stored in them. It is necessary to indicate that, although the sizes of the files used in this work are smaller than those that can be worked in large companies such as Google, among others, it does help to have an idea of the behaviour that can have if it were to be used in larger environments. It should be added that, as can be seen in the tests, Apache Cassandra can be used in some cases as a File System, but, on the other hand, it is not up to those systems that are truly optimized to handle these types of data.

In this way, it can be concluded that the main objective for determining whether a Database can be used for the development of a Distributed and Parallel File System has been successfully achieved.

G.2. **Future work**

After the present project has been completed, a series of works or future lines are shown, which may serve to improve this project:

- Full integration of the interface developed in MPI-IO.
- Evaluation of the performance of the interface developed in a cloud environment, such as the environment offered by Amazon, called Amazon EC2.
- Integration of the interface developed in Mimir, which is a library that allows users to program Map and Reduce functions within MPI environments.
- Make a module to include the library developed within the Linux kernel using FUSE (Filesystem in User Space).

Bibliografía

[1] K Shvachko, H. Kuang, S Radia y R Chansler, "The Hadoop Distributed File System", eng, en *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, 2010, pp. 1-10.

[2] A. Ergüzen y M. ünver, "Developing a file system structure to solve healthy big data storage and archiving problems using a distributed file system", *Applied Sciences (Switzerland)*, vol. 8, n.º 6, <xocs:first xmlns:xocs=/>, 2018, ID: TN_scopus2-s2.0-85048127631. DOI: 10.3390/app8060913.

[3] D. E. Holmes, *Big data : una breve introducción*, spa. Barcelona: : Antoni Bosch, 2018.

[4] A. Barton, "Big Data", *Journal Of Nursing Education*, vol. 55, n.º 3, pp. 123-124, 2016, ID: TN_wos000377378600001. DOI: 10.3928/01484834-20160216-01.

[5] L. Page. (). Google, [En línea]. Disponible en: https://www.google.es/ (Acceso: 31-03-2019).

[6] J. M. Hoya Quecedo, *Sistema de ficheros para GNU/Linux con monitorización de operaciones*, spa, 2017.

[7] S. Ghemawat, H. Gobioff y S. T Leung, "The google file system", *Operating Systems Review (ACM)*, vol. 37, n.º 5, pp. 29-43, 2003,
ID: TN_scopus2-s2.0-21644437974. DOI: 10.1145/1165389.945450.

[8] *Big data: The next Google*, ID: RS_6002808368bigdatathenextgoogle, 2008.

[9] L. Page. (). Google Drive, [En línea]. Disponible en: https://www.google.com/intl/es_ALL/drive/ (Acceso: 31-03-2019).

[10] B. Behlendorf. (). Apache Software Foundation, [En línea]. Disponible en: https://www.apache.org/ (Acceso: 31-03-2019).

[11] A. Lakshman y P. Malik. (). Apache Cassandra, [En línea]. Disponible en: http://cassandra.apache.org/ (Acceso: 01-09-2018).

[12] A. Agea Herradón, *Diseño e implementacion de un sistema de ficheros sobre una base de datos NoSQL*, spa, 2017.

[13] D. Axmark, A. Larsson y M. Widenius. (). MySQL, [En línea]. Disponible en: https://www.mysql.com/ (Acceso: 15-10-2018).

[14] L. Ellison. (). Oracle Corporation, [En línea]. Disponible en: https://www.oracle.com/es/corporate/ (Acceso: 15-10-2018).

[15] J. Ellis. (). Datastax, [En línea]. Disponible en: https://www.datastax.com/ (Acceso: 15-10-2018).

[16] C. A. I. Staff, *IEEE Guide to the POSIX Open System Environment (OSE)(ANSI), 1003.0-1995*. Place of publication not identified IEEE, ID: 34UC3M_ALMA51238619780004213.

[17] D. Nagle, "MPI – The Complete Reference, Vol. 1, The MPI Core, 2nd ed., Scientific and Engineering Computation Series, by Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker and Jack Dongarra", eng, *Scientific Programming*, vol. 13, n.º 1, pp. 57-63, 2005. [En línea]. Disponible en: https://doaj.org/article/606eb7e72fce49889a2287589229e724.

[18] S. Sehrish y J. Wang, "Reduced Function Set Abstraction (RFSA) for MPI-IO", eng, *The Journal of Supercomputing*, vol. 59, n.º 1, pp. 131-146, 2012.

[19] F. J. R. Duro, *Diseño e implementación de un sistema de ficheros distribuido basado en Memcached*, ID: 34UC3M_DSP10016/13003, 2012.

[20] G. F. Coulouris, *Sistemas distribuidos : conceptos y diseño*, 1ª ed. en español, 3ª ed. en ingleés. Madrid etc.: Madrid etc. : Addison Wesley, 2001, ID: 34UC3M_ALMA21173546420004213.

[21] L. M. S. García, *Sistema de ficheros paralelo escalable para entornos "cluster"*, ID: TN_cbuc_tes10803/14532, 2011.

[22] M. Widenius. (). MariaDB, [En línea]. Disponible en: https://mariadb.org/ (Acceso: 20-10-2018).

[23] V. N. Gudivada, R. Baeza-Yates y V. V. Raghavan, "Big Data: Promises and Problems", *Computer*, vol. 48, n.º 3, pp. 20-23, 2015,
ID: TN_ieee_s7063181. DOI: 10.1109/MC.2015.62.

[24] D. Cutting y M. Cafarella. (). HDFS, [En línea]. Disponible en: https://hadoop.apache.org/docs/current1/hdfs_design.html (Acceso: 20-10-2018).

[25] D. Dev y R. Patgiri, *Performance evaluation of HDFS in big data management*, ID: TN_ieee_s7045330, 2014. DOI: 10.1109/ICHPCA.2014.7045330.

[26] J. Dean y S. Ghemawat, "MapReduce: simplified data processing on large clusters", *Communications of the ACM*, vol. 51, n.º 1, pp. 107-113, 2008,
ID: TN_acm1327492. DOI: 10.1145/1327452.1327492.

[27] M. Madison, M. Barnhill, C. Napier y J. Godin, "NoSQL Database Technologies", *Journal of International Technology and Information Management*, vol. 24, n.º 1, p. I, 2015, ID: TN_proquest1757727794.

[28] N. Neeraj, *Mastering Apache Cassandra*. Birmingham : Packt Publishing, 2013, ID: 34UC3M_ALMA51195427770004213.

[29] M. J. Donahoo y G. D. Speegle, *SQL*, eng, 1.ª ed. Morgan Kaufmann, 2010.

[30] S. Gilbert y N. Lynch, "Perspectives on the CAP Theorem", *Computer*, vol. 45, n.º 2, pp. 30-36, 2012, ID: TN_ieee_s6122006. DOI: 10.1109/MC.2011.389.

[31] A. Singh, *Instant Cassandra Query Language*. Birmingham: Birmingham: Packt Publishing Ltd, 2013, ID: TN_pq_ebook_centralEBC1420552.

[32] V. Mishra, *Beginning Apache Cassandra Development*. Berkeley, CA : Apress : Imprint: Apress, 2014, Includes bibliographical references at the end of each chapters and index.; ID: 34UC3M_ALMA51208745750004213.

[33] *IEEE Std 1003.1-2008 (Revision of IEEE Std 1003.1-2004): IEEE Standard for Information Technology- Portable Operating System Interface (POSIX) Base Specifications, Issue 7*. 2008, ID: 34UC3M_ALMA51238619630004213.

[34] D. S. Ray, *UNIX*. Place of publication not identified Peachpit Press, 1998, ID: 34UC3M_ALMA51231672340004213.

[35] A. d. Frutos Ballesteros, *Implementacion de la interfaz de E/S MPI-IO para el sistema de fichero paralelo Expand*, spa. Leganeés: : A. Frutos, 2002.

[36] (). Página principal de MPICH, [En línea]. Disponible en: https://www.boe.es/buscar/act.php?id=BOE-A-1992-28740 (Acceso: 01-05-2019).

[37] R. Thakur, E. Lusk y W. Gropp, *Users guide for ROMIO: A high-performance, portable MPI-IO implementation*, eng, 1997.

[38] R Bordawekar, J Del Rosario y A Choudhary, "Design and Evaluation of primitives for Parallel I/O", eng, en *Proceedings of the 1993 ACM/IEEE conference on supercomputing*, ép. Supercomputing '93, ACM, 1993, pp. 452-461.

[39] R. Thakur, W. Gropp y E. Lusk, "An abstract-device interface for implementing portable parallel-I/O interfaces", eng, *6. symposium on the frontiers of massively parallel computation, Anapolis*, 1996.

[40] (). Driver Apache Cassandra C, [En línea]. Disponible en: https://docs.datastax.com/en/developer/cpp-driver/2.11/ (Acceso: 15-09-2018).

[41] *IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications*. IEEE, 1998, ID: 34UC3M_ALMA51229516760004213.

[42] J. B. Cruz, *Trazabilidad de requisitos no funcionales*. Leganeés: Leganeés : J. Bermudo, 2006, ID: 34UC3M_ALMA21162569340004213.

[43] (). Licencia de Apache, [En línea]. Disponible en: http://www.apache.org/licenses/ (Acceso: 03-03-2019).

[44] (). Licencia Driver Datastax, [En línea]. Disponible en: http://www.apache.org/licenses/LICENSE-2.0 (Acceso: 03-03-2019).

[45] (). Documentación del protocolo de Apache Cassandra, [En línea]. Disponible en: https://github.com/datastax/cpp-driver (Acceso: 15-09-2018).

[46] R. S. Pressman, *Ingenieriéa del software : un enfoque praéctico*, spa, 7ª ed. Meéxico D.F. [etc.: : McGraw-Hill Interamericana, 2010.

[47] M. F. Smith, *Software prototyping : adoption, practice and management*, eng, ép. McGraw-Hill software engineering series. London [etc.: : McGraw-Hill, 1991.

[48] B. W Boehm, "A spiral model of software development and enhancement", eng, *Computer*, vol. 21, n.º 5, pp. 61-72, 1988.

[49] (1992). Ley del Impuesto Valor Añadido, [En línea]. Disponible en: https://www.boe.es/buscar/act.php?id=BOE-A-1992-28740 (Acceso: 15-04-2019).